

Subgraph Query Examples

This doc will teach you how to query Uniswap V3 analytics by writing GraphQL queries on the subgraph. You can fetch data points like :

- [collected fees for a position](#)
- [current liquidity](#) of a pool
- [volume on a certain day](#)

and much more. Below are some example queries. To run a query copy and paste it into the [v3 explorer](#) to get fresh data.

Global Data

Global data refers to data points about the Uniswap v3 protocol as a whole. Some examples of global data points are total value locked in the protocol, total pools deployed, or total transaction counts. Thus, to query global data you must pass in the Uniswap V3 Factory address `0x1F98431c8aD98523631AE4a59f267346ea31F984` and select the desired fields. Reference the full [factory schema](#) to see all possible fields.

Current Global Data

An example querying total pool count, transaction count, and total volume in USD and ETH:

```
{
  factory(id: "0x1F98431c8aD98523631AE4a59f267346ea31F984") {
    poolCount
    txCount
    totalVolumeUSD
    totalVolumeETH
  }
}
```

Historical Global Data

You can also query historical data by specifying a block number.

```
{
  factory(id: "0x1F98431c8aD98523631AE4a59f267346ea31F984", block: {number: 13380584}) {
    poolCount
    txCount
    totalVolumeUSD
    totalVolumeETH
  }
}
```

Pool Data

To get data about a certain pool, pass in the pool address. Reference the full [pool schema](#) and adjust the query fields to retrieve the data points you want.

General Pool Query

The query below returns the feeTier, spot price, and liquidity for the ETH-USDC pool.

```
{
  pool(id: "0x8ad599c3a0ff1de082011efddc58f1908eb6e6d8") {
    tick
    token0 {
      symbol
      id
      decimals
    }
    token1 {
      symbol
      id
      decimals
    }
    feeTier
    sqrtPrice
    liquidity
  }
}
```

All Possible Pools

The maximum items you can query at once is 1000. Thus to get all possible pools, you can iterate using the skip variable. To get pools beyond the first 1000 you can also set the skip as shown below.

Skipping First 1000 Pools

This query sets the skip value and returns the first 10 responses after the first 1000.

```
{
  pools(first:10, skip:1000) {
    id
    token0 {
      id
      symbol
    }
    token1 {
      id
      symbol
    }
  }
}
```

Creating a Skip Variable

This next query sets a skip variable. In your language and environment of choice you can then iterate through a loop, query to get 1000 pools each time, and continually adjust skip by 1000 until all pool responses are returned.

Check out [this example](#) from our interface for poolDayData that does something similar.

Note: This query will not work in the graph explorer and more resembles the structure of a query you'd pass to some graphql middleware like Apollo.

```
query pools( $skip: Int!) {
  pools(
    first: 1000
    skip: $skip
    orderDirection: asc
  ) {
    id
    sqrtPrice
    token0 {
      id
    }
    token1 {
      id
    }
  }
}
```

Most Liquid Pools

Retrieve the top 1000 most liquid pools. You can use this similar set up to orderBy other variables like number of swaps or volume.

```
{
  pools(first: 1000, orderBy: liquidity, orderDirection: desc) {
    id
  }
}
```

Pool Daily Aggregated

This query returns daily aggregated data for the first 10 days since the given timestamp for the UNI-ETH pool.

```
{
  poolDayDatas(first: 10, orderBy: date, where: {
    pool: "0x1d42064fc4beb5f8aaaf85f4617ae8b3b5b8bd801",
    date_gt: 1633642435
  }) {
    date
    liquidity
    sqrtPrice
  }
}
```

```
    token0Price
    token1Price
    volumeToken0
    volumeToken1
  }
}
```

Swap Data

General Swap Data

To query data about a particular swap, input the transaction hash + "#" + the index in the swaps the transaction array. This is the reference for the full [swap schema](#).

This query fetches data about the sender, receiver, amounts, transaction data, and timestamp for a particular swap.

```
{
  swap(id: "0x000007e1111cbd97f74cf6eeaa2879a5b02020f26960ac06f4af0f9395372b64#66785") {
    sender
    recipient
    amount0
    amount1
    transaction {
      id
      blockNumber
      gasUsed
      gasPrice
    }
    timestamp
    token0 {
      id
      symbol
    }
    token1 {
      id
      symbol
    }
  }
}
```

Recent Swaps Within a Pool

You can set the `where` field to filter swap data by pool address. This example fetches data about multiple swaps for the USDC-USDT pool, ordered by timestamp.

```
{
  swaps(orderBy: timestamp, orderDirection: desc, where: {
    pool: "0x7850e59e0c01ea06df3af3d20ac7b0003275d4bf"
  }) {
    pool {
      token0 {
        id
        symbol
      }
      token1 {
        id
        symbol
      }
    }
    sender
    recipient
    amount0
    amount1
  }
}
```

Token Data

Input the the token contract address to fetch token data. Any token that exists in at least one Uniswap V3 pool can be queried. The output will aggregate data across all v3 pools that include the token.

General Token Data

This queries the decimals, symbol, name, pool count, and volume in USD for the UNI token. Reference the full [token schema](#) for all possible fields you can query.

```
{  
  token(id:"0x1f9840a85d5af5bf1d1762f925bdaddc4201f984") {  
    symbol  
    name  
    decimals  
    volumeUSD  
    poolCount  
  }  
}
```

Token Daily Aggregated

You can fetch aggregate data about a specific token over a 24-hour period. This query gets 10-days of the 24-hour volume data for the UNI token ordered from oldest to newest.

```
{  
  tokenDayDatas(first: 10, where: {token: "0x1f9840a85d5af5bf1d1762f925bdaddc4201f984"}, orderBy: date,  
  orderDirection: asc) {  
    date  
    token {  
      id  
      symbol  
    }  
    volumeUSD  
  }  
}
```

All Tokens

Similar to retrieving all pools, you can fetch all tokens by using skip. Note: This query will not work in the graph sandbox and more resembles the structure of a query you'd pass to some graphql middleware like Apollo.

```
query tokens($skip: Int!) {  
  tokens(first: 1000, skip: $skip) {  
    id  
    symbol  
    name  
  }  
}
```

Position Data

General Position Data

To get data about a specific position, input the NFT tokenId. This queries the collected fees for token0 and token1 and current liquidity for the position with tokedId 3. Reference the full [position schema](#) to see all fields.

```
{  
  position(id:3) {  
    id  
    collectedFeesToken0  
    collectedFeesToken1  
    liquidity  
    token0 {  
      id  
      symbol  
    }  
    token1 {  
      id  
      symbol  
    }  
  }  
}
```

Contribute

There are many more queries you can do with the Uniswap v3 subgraph including data related to ticks, mints, positions, and burns. Once again you can reference the full schema [here](#). If you'd like to suggest more example queries to showcase, feel free to drop some suggestions in [discord](#) under #dev-chat or [contribute](#) your own queries by submitting a pull request to the docs repo.

id: overview sidebar_position: 1 title: Overview

The Uniswap Subgraph

Uniswap uses multiple [subgraphs](#) for indexing and organizing data from the Uniswap smart contracts. These subgraphs are hosted on The Graph hosted service and can be used to query Uniswap data.

Versions and Production Endpoints

Each version of Uniswap has its own dedicated subgraph, and governance contracts have a dedicated subgraph as well.

Each subgraph has a dedicated endpoint for querying data, as well as a page on [The Graph explorer](#) the exposes the schema and available fields to query.

V3

- Explorer Page: <https://thegraph.com/explorer/subgraph/uniswap/uniswap-v3>
- Graphql Endpoint: <https://api.thegraph.com/subgraphs/name/uniswap/uniswap-v3>
- Code: <https://github.com/Uniswap/uniswap-v3-subgraph>

Governance

- Explorer Page: <https://thegraph.com/explorer/subgraph/ianlapham/governance-tracking>
- Graphql Endpoint: <https://api.thegraph.com/subgraphs/name/ianlapham/governance-tracking>
- Code: <https://github.com/ianlapham/uniswap-governance-subgraph>

V2

- Explorer Page: <https://thegraph.com/explorer/subgraph/ianlapham/uniswapv2>
- Graphql Endpoint: <https://api.thegraph.com/subgraphs/name/ianlapham/uniswapv2>
- Code: <https://github.com/Uniswap/uniswap-v2-subgraph>

V1

- Explorer Page: <https://thegraph.com/explorer/subgraph/ianlapham/uniswap>
- Graphql Endpoint: <https://api.thegraph.com/subgraphs/name/ianlapham/uniswap>
- Code: <https://github.com/graphprotocol/uniswap-subgraph>

id: glossary title: Glossary sidebar_position: 6

Automated Market Maker

An automated market maker is a smart contract on Ethereum that holds liquidity reserves. Users can trade against these reserves at prices determined by a fixed formula. Anyone may contribute liquidity to these smart contracts, earning pro-rata trading fees in return.

Asset

While a digital asset can take many forms, the Uniswap Protocol supports ERC-20 token pairs, and represents a position in the form of an NFT (ERC-721).

Concentrated Liquidity

Liquidity that is allocated within a determined price range.

Constant Product Formula

The automated market making algorithm used by Uniswap. In v1 and v2, this was $x \cdot y = k$.

Core

Smart contracts that are considered foundational, and are essential for Uniswap to exist. Upgrading to a new version of core would require deploying an entirely new set of smart contracts on Ethereum and would be considered a new version of the Uniswap Protocol.

ERC20

ERC20 tokens are fungible tokens on Ethereum. Uniswap supports all standard ERC20 implementations.

Factory

A smart contract that deploys a unique smart contract for any ERC20/ERC20 trading pair.

Flash Swap

A trade that uses the tokens purchased before paying for them.

Invariant

The "k" value in the constant product formula $X*Y=K$

Liquidity Provider / "LP"

A liquidity provider is someone who deposits ERC20 tokens into a given liquidity pool. Liquidity providers take on price risk and are compensated with trading fees.

Liquidity

Digital assets that are stored in a Uniswap pool contract, and are able to be traded against by traders.

Mid Price

The price between the available buy and sell prices. In Uniswap V1 and V2, this is the ratio of the two ERC20 token reserves. In V3, this is the ratio of the two ERC20 token reserves available within the current active tick.

Observation

An instance of historical price and liquidity data of a given pair.

Pair

A smart contract deployed from a Uniswap V1 or V2 factory contract that enables trading between two ERC20 tokens. Pair contracts are now called Pools in V3.

Periphery

External smart contracts that are useful, but not required for Uniswap to exist. New periphery contracts can always be deployed without migrating liquidity.

Pool

A contract deployed by the V3 factory that pairs two ERC-20 assets. Different pools may have different fees despite containing the same token pair. Pools were previously called Pairs before the introduction of multiple fee options.

Position

An instance of liquidity defined by upper and lower tick. And the amount of liquidity contained therein.

Price Impact

The difference between the mid-price and the execution price of a trade.

Protocol Fees

Fees that are rewarded to the protocol itself, rather than to liquidity providers.

Range

Any interval between two ticks of any distance.

Range Order

An approximation of a limit order, in which a single asset is provided as liquidity across a specified range, and is continuously swapped to the destination address as the spot price crosses the range.

Reserves

The liquidity available within a pair. This was more commonly referenced before concentrated liquidity was introduced.

Slippage

The amount the price moves in a trading pair between when a transaction is submitted and when it is executed.

Spot Price

The current price of a token relative to another within a given pair.

Swap Fees

The fees collected upon swapping which are rewarded to liquidity providers.

Tick Interval

The price space between two nearest ticks.

Tick

The boundaries between discrete areas in price space.

id: overview title: Overview

Code

[governance](#)

Documentation

For reference material on the Uniswap Governance system please see [Governance Reference](#).

UNI Address

`UNI` is deployed at `0x1f9840a85d5aF5bf1D1762F925BDADDc4201F984` on the Ethereum [mainnet](#), and the [Ropsten](#), [Rinkeby](#), [Görli](#), and [Kovan](#) testnets. It was built from commit [ab22c08](#).

ABI

```
import Uni from '@uniswap/governance/build/Uni.json'
```

<https://unpkg.com/@uniswap/governance@1.0.2/build/Uni.json>

Timelock

`Timelock` is deployed at `0x1a9C8182C09F50C8318d769245beA52c32BE35BC` on the Ethereum [mainnet](#), and the [Ropsten](#), [Rinkeby](#), [Görli](#), and [Kovan](#) testnets. It was built from commit [ab22c08](#).

ABI

```
import Timelock from '@uniswap/governance/build/Timelock.json'
```

<https://unpkg.com/@uniswap/governance@1.0.2/build/Timelock.json>

GovernorAlpha (Deprecated)

`GovernorAlpha` is deployed at `0x5e4be8Bc9637f0EAA1A755019e06A68ce081D58F` on the Ethereum [mainnet](#), and the [Ropsten](#), [Rinkeby](#), [Görli](#), and [Kovan](#) testnets. It was built from commit [ab22c08](#).

ABI

The `GovernorAlpha` ABI is viewable on [Etherscan](#), as well as via an [npm package](#).

<https://unpkg.com/@uniswap/governance@1.0.2/build/GovernorAlpha.json>

```
import GovernorAlpha from '@uniswap/governance/build/GovernorAlpha.json'
```

GovernorAlpha v2 (Deprecated)

GovernorAlpha v2 is deployed at 0xC4e172459f1E7939D522503B81AFaaC1014CE6F6 on the Ethereum [mainnet](#).

ABI

The GovernorAlpha v2 ABI is viewable on [Etherscan](#)

GovernorBravo (Current)

GovernorBravo is deployed at 0x408ED6354d4973f66138C91495F2f2FCbd8724C3 on the Ethereum [mainnet](#).

ABI

The Governor Bravo ABI can be found on [Etherscan](#).

Miscellaneous Addresses

The following addresses only exist on the Ethereum mainnet.

The UNI merkle distributor address is 0x090D4613473dEE047c3f2706764f49E0821D256e .

The staking rewards factory address is 0x3032Ab3Fa8C01d786D29dAdE018d7f2017918e12 .

The four staking rewards addresses are:

```
0x6c3e4cb2e96b01f4b866965a91ed4437839a121a
0x7fba4b8dc5e7616e59622806932dbea72537a56b
0xa1484c3aa22a66c62b77e0ae78e15258bd0cb711
0xca35e32e7926b96a9988f61d510e038108d8068e
```

The four year-long vesting contract addresses are:

```
0x4750c43867ef5f89869132eccf19b9b6c4286e1a
0xe3953d9d317b834592ab58ab2c7a6ad22b54075d
0x4b4e140d1f131fdad6fb59c13af796fd194e4135
0x3d30b1ab88d487b0f3061f40de76845bec3f1e94
```

The feeToSetterVester address is 0x18e433c7Bf8A2E1d0197CE5d8f9AFAda1A771360 .

The feeTo address is 0xDaF819c2437a82f9e01f6586207ebF961a7f0970 .

id: process title: Process

This document is a living document which represents the current process guidelines for developing and advancing Uniswap Governance Proposals.

Process

Several governance venues are available to Uniswap governance, each serving its own particular purpose.

1. [gov.uniswap.org](#)

gov.uniswap.org is a Discourse-hosted forum for governance-related discussion. Community members must register for an account before sharing or liking posts. New members are required to enter 4 topics and read 15 posts over the course of 10 minutes before they are permitted to post themselves.

2. [Snapshot](#)

Snapshot is a simple voting interface that allows users to signal sentiment off-chain. Votes on snapshot are weighted by the number of UNI delegated to the address used to vote.

3. [Governance Portal](#)

The formal governance portal can be accessed directly through the Uniswap app interface. Votes are delegated and cast through the portal.

Below we outline a preliminary draft for the Uniswap governance process, detailing exactly where these venues fit in. These processes are subject to change according to feedback from the Uniswap community.

Phase 1: Temperature Check — Discourse/Snapshot

The purpose of the Temperature Check is to determine if there is sufficient will to make changes to the status quo.

To create a Temperature Check:

1. Ask a general, non-biased question to the community on gov.uniswap.org about a potential change (example: "Should Uniswap governance add liquidity mining for XYZ token?"). Forum posts should be labeled as follows: "Temperature Check - [Your Title Here]". The forum post should include a link to the associated Snapshot poll.
2. Voters use Snapshot to indicate their interest in bringing it forward to the next stage. Snapshot poll lengths should be set to 2 days.

That's it! You've just started the process of gaining support for a proposal. At the end of the 2 days, a majority vote with a 25k UNI yes-vote threshold wins.

If the Temperature check does not suggest a change from the status quo, the topic will be closed on the governance site. If the Temperature Check does suggest a change, proceed to Stage 2: Consensus Check.

Phase 2: Consensus Check — Discourse/Snapshot

The purpose of the Consensus Check is to establish formal discussion around a potential proposal.

To create a Consensus Check:

1. Use feedback from the Temperature Check post and create a new Snapshot poll which covers the options which have gained support. This poll can either be binary or multiple choice but you are required to include the option "Make no change" or its equivalent. Set the poll duration to 5 days.
2. Create a new topic in the Proposal Discussion category on gov.uniswap.org titled "Consensus Check — [Your Title Here]". This will alert the community that this topic has already passed Temperature Check. Any topics beginning with Consensus Check that have not passed Temperature Check will immediately be removed by moderators. Make sure that the discussion thread links to the new Snapshot poll and the Temperature Check thread.
3. Reach out to your network to build support for the proposal. Discuss the proposal and actively solicit delegates to vote on it. Be willing to respond to questions on the Consensus Check topic. Share your view point, although try to remain as impartial as possible.

At the end of 5 days, whichever option has the majority of votes wins, and can be included in a governance proposal for Stage 3. A 50k UNI yes-vote quorum is required for the Consensus Check to pass.

If the option "Make no change" wins, the Consensus Check topic will be closed by the moderators.

Phase 3: Governance Proposal — Governance Portal

Phase 3 — Governance Proposal — is the final step of the governance process. The proposal should be based on the winning outcome from the Consensus Check and can consist of one or multiple actions, up to a maximum of 10 actions per proposal.

To create a Governance Proposal:

1. Write the code for your proposal, which will be voted on through the Governance Portal. More resources can be found [here](#). All proposed code should be audited by a professional auditor. This auditing process may be paid or reimbursed by the community treasury.
2. Ensure that you have at least 2.5 million UNI delegated to your address in order to submit a proposal, or find someone who has enough UNI to meet the proposal threshold to propose on your behalf.
3. Create a topic in the Proposal Discussion category on gov.uniswap.org titled "Governance Proposal — [Your Title Here]" and link to any relevant Snapshot polls/discussion threads as well as the code audit report. Topics that begin with "Governance Proposal" that have not successfully passed through the Temperature Check and Consensus Check stages will be removed by moderators.
4. Call the propose() function of the Governor Bravo to deploy your proposal.

Once the propose() function has been called, a two day voting delay will start. After voting delay is finished a seven day voting period begins. Ongoing discussion can take place in the gov.uniswap.org forum. If the proposal passes successfully, a two day timelock will follow before the proposed code is executed.

Soft governance

The process described above lays out a structure for those wishing to host a formal vote around a particular issue.

However, governing this system also requires a degree of "meta governance", discussions that inform the direction of and the implementation processes behind policy but which don't qualify as policy themselves.

The community may discuss new ideas and strategies for governance — including changes to the three-step process outlined above — in the "Governance-Meta" category. On-chain voting is not necessary to make updates to off-chain processes.

Governance Glossary

- **UNI:** An ERC-20 token that designates the weight of a user's voting rights. The more UNI a user has in their wallet, the more weight their delegation or vote on a proposal holds.
- **Delegation:** UNI holders cannot vote or create proposals until they delegate their voting rights to an address. Delegation can be given to one address at a time, including the holder's own address. Note that delegation does not lock tokens; it simply adds votes to the chosen delegation address.
- **Proposal:** A proposal is executable code that modifies the governance contract or treasury and how they work. In order to create a proposal, a user must have at least 0.25% (2.5M UNI) of all UNI delegated to their address. Proposals are stored in the "proposals" mapping of the Governor smart contract. All proposals are subject to a 7-day voting period. If the proposer does not maintain their vote weight balance throughout the voting period, the proposal may be canceled by anyone.
- **Quorum:** In order for a vote to pass, it must achieve quorum of 4% of all UNI (40M) voting in the affirmative. The purpose of the quorum is to ensure that the only measures that pass have adequate voter participation.
- **Voting:** Users can vote for or against single proposals once they have voting rights delegated to their address. Votes can be cast while a proposal is in the "Active" state. Votes can be submitted immediately using "castVote" or submitted later with "castVoteBySig" (For more info on castVoteBySig and offline signatures, see EIP-712). If the majority of votes (and a 4% quorum of UNI) vote for a proposal, the proposal may be queued in the Timelock.
- **Voting Period:** Once a proposal has been put forward, Uniswap community members will have a seven day period (the Voting Period) to cast their votes.
- **Timelock:** All governance and other administrative actions are required to sit in the Timelock for a minimum of 2 days, after which they can be implemented.

id: guide-to-voting title: Beginners Guide to Voting

This guide contains everything you need to start voting in Uniswap Governance.

In order to participate you will need:

- [UNI Tokens](#)
- ETH for transaction costs
- A browser with [Metamask](#) installed

The governance [process](#) begins in the [Governance Forum](#), where you can find proposals under consideration, gather information about community sentiment, and engage with the community.

Once a given proposal has made it through the proposal process and is ready for voting, it will appear in the Uniswap [voting dashboard](#) - where you can view all current and former Uniswap proposals.

If a proposal is currently live for voting, it will say `active` next to the title. Clicking the proposal will show all the necessary information, documentation, and discussion needed for a voter to make an informed decision.

Once a proposal has reached the voting stage it represents real, executable code which will alter the functionality of Uniswap Governance or anything under its jurisdiction - proper care should be taken to ensure that the code represented in the proposal has been audited and is found to be in good faith.

Delegation

UNI is a tradable asset and functions like most other standard ERC20 tokens, except it has a deeper power as a voting mechanism. In order for UNI to be used as a vote, the owner must first go through the delegation process. Delegating UNI binds the voting power of your tokens to an address so it may be used to vote. This address could be yourself, or a trusted party who you believe will vote in the best interest of Uniswap Governance.

A democratic consensus, in our process called "quorum", is determined by the percentage of UNI tokens in favor of, or against, a proposal. 1% of all UNI must be cast in favor to submit a proposal, and 4% in order to pass a vote.

To delegate your UNI tokens and enact their voting power, visit the [Uniswap voting dashboard](#) and click the button that says "Unlock Voting".

Once you click this button, you will see a screen that gives you the option to self delegate, or add a delegate address. If you wish to delegate your UNI voting power to your own address, click "Self Delegate".

When you click "Self Delegate", a transaction will pop up in Metamask. If this doesn't happen, double check that metamask is connected to app.uniswap.org, turn off any popup blockers, and try again. Click confirm, and once the transaction has processed, you will see that the voting

dashboard homepage has changed to show the number of votes you have, and "Delegated to: Self".

If you wish to delegate your voting power to another party, choose "Add Delegate" and enter the ethereum address of your chosen voting party.

An important note: much like voter registration in a larger democracy, for UNI to be used in a vote it must be delegated before both the voting period and the preceding proposal period. This means if you want your vote to count, you must delegate it in anticipation of any proposal you may be interested in.

If you are unsure of how best to vote and are interested in delegating your UNI voting power to another party, you can visit the [Delegation Pitch](#) section of the governance forum. Here parties participating in Uniswap Governance pitch their platform and voting agenda for users to read and discuss.

Voting

If you have successfully self delegated and there is an active proposal, you are ready to vote in Uniswap Governance.

To cast your vote, navigate to the proposals page and click on an active proposal.

After reviewing the attached details and deciding your opinion, choose "Vote For", or "Vote Against".

Once you've chosen, a window will pop up allowing you to execute the vote.

When you click to cast your vote, metamask will pop up asking you to confirm your transaction. Click "submit", wait a bit, and check that the transaction has been confirmed.

That's it! Once your transaction has been confirmed, you will have cast your vote and participated in Uniswap Governance.

id: glossary title: Glossary

- **UNI:** An ERC-20 token that designates the weight of a user's voting rights. The more UNI a user has in their wallet, the more weight their delegation or vote on a proposal holds.
- **Delegation:** UNI holders cannot vote or create proposals until they delegate their voting rights to an address. Delegation can be given to one address at a time, including the holder's own address. Note that delegation does not lock tokens; it simply adds votes to the chosen delegation address.
- **Proposal:** A proposal is executable code that modifies the governance contract or treasury and how they work. In order to create a proposal, a user must have at least 0.25% (2.5M UNI) of all UNI delegated to their address. Proposals are stored in the "proposals" mapping of the Governor smart contract. All proposals are subject to a 7-day voting period. If the proposer does not maintain their vote weight balance throughout the voting period, the proposal may be canceled by anyone.
- **Quorum:** In order for a vote to pass, it must achieve quorum of 4% of all UNI (40M) voting in the affirmative. The purpose of the quorum is to ensure that the only measures that pass have adequate voter participation.
- **Voting:** Users can vote for or against single proposals once they have voting rights delegated to their address. Votes can be cast while a proposal is in the "Active" state. Votes can be submitted immediately using "castVote" or submitted later with "castVoteBySig" (For more info on castVoteBySig and offline signatures, see EIP-712). If the majority of votes (and a 4% quorum of UNI) vote for a proposal, the proposal may be queued in the Timelock.
- **Voting Period:** Once a proposal has been put forward, Uniswap community members will have a seven day period (the Voting Period) to cast their votes.
- **Timelock:** All governance and other administrative actions are required to sit in the Timelock for a minimum of 2 days, after which they can be implemented.

id: adversarial-circumstances title: Adversarial Circumstances

This document explores some adversarial circumstances which Uniswap Governance may encounter in the future. Its goal is to help those interested in Uniswap Governance understand the reasoning behind some of its design, its limitations, and potential avenues for growth.

Scenario 1

A good faith proposal is brought to vote but is found to have an exploitable edge case. A bad faith actor uses a series of DeFi leveraging strategies to quickly buy enough UNI during the voting period to sway the vote in favor of the proposal, passing it and exploiting the vulnerability.

Circumvention

UNI voting power must be delegated to an address either entirely before a proposal has been submitted or during the proposal delay period. For now, the proposal delay is set to one block, which is about 15 seconds. A proposal delay of one block leaves no opportunity for a third party to find an exploitable edge case and opportunistically purchase uni, self delegate and sway the vote.

In the future, Uniswap Governance may vote to increase the proposal delay. While there are obvious benefits to an increased proposal delay, it may introduce some potential adverse outcomes such as opportunistic edge case exploitation.

Scenario 2

A bad faith proposal is crafted and submitted to vote, which is unambiguously not in the best interest of Uniswap Governance. Multiple parties collude ahead of time to corner the UNI market to force the proposal through, gain access to the UNI reserves, and drain the funds.

Circumvention

Since UNI is a freely tradable asset, anyone can attempt a governance takeover via market buying. That said, to force-pass a bad faith vote would require a minimum of 40 million UNI. If not outright impossible, this amount would be prohibitively expensive and likely cost more when accounting for price fluctuation than the net gain from the attack.

If a group somehow achieved a bad faith takeover, Timelock's delay would give affected agents time to withdraw their assets from the protocol. This would also be an opportunity to fork the protocol, a path that would likely be taken by the remaining good-faith actors.

Scenario 3

A single party uses a flash loan to push through a proposal, potentially creating a pseudo-DDOS attack by spamming governance with proposals and preventing effective use.

Circumvention

A delegated balance of 2.5 million UNI is required to submit a vote, but the balance check is set exactly one block in the past. This prevents any flash loan proposals from being created, as flash loans cannot execute outside of a single block.

The proposer must also maintain a minimum balance of 2.5 million UNI throughout the voting period, or anyone may cancel the proposal. This balance maintenance check prevents many highly leveraged proposal techniques that may span several blocks.

Scenario 4

A bad faith proposal is created, which will genuinely incentivize bad faith voting.

Example: "The treasury will be drained. Any votes in favor will be sent the balance of the treasury. Any votes opposed will be locked from the funds of the treasury."

Circumvention

No mechanism explicitly prevents this type of scenario, but market forces disincentivize it.

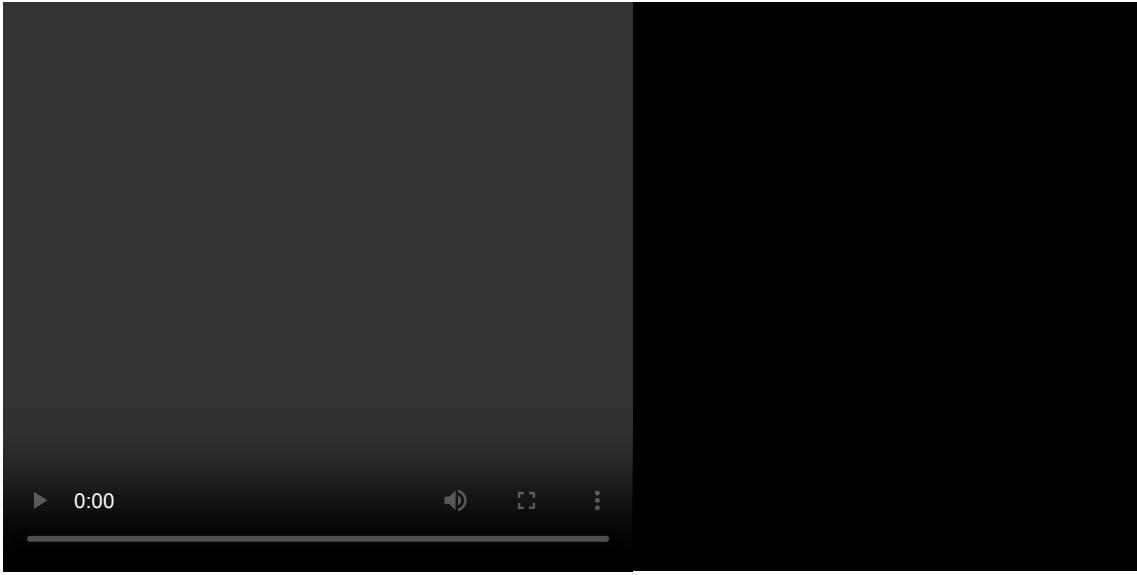
Because the treasury is comprised of UNI tokens exclusively, the market would react appropriately if a vote were to pass that would jeopardize the economic viability of Uniswap Governance and the UNI token. By the time the vote would pass, UNI's price would have fallen so low as to make the attack fruitless.

UNI acting as the only asset of the governance treasury disincentivizes this form of bad faith voting. Uniswap Governance may choose in the future to diversify governance assets. While there are many benefits to this path, some fringe possibilities such as incentivized bad faith voting may appear.

id: changelog title: Changelog

Date	Abstract	Transaction	Sybil Reference
June 12, 2021	Reduce the UNI proposal submission threshold to 2.5M	Etherscan	https://sybil.org/#/proposals/uniswap/4
August 26, 2021	Upgrade Governance Contract to Compound's Governor Bravo	Etherscan	https://sybil.org/#/proposals/uniswap/8

id: concentrated-liquidity title: Concentrated Liquidity sidebar_position: 1



Introduction

The defining idea of Uniswap v3 is concentrated liquidity: liquidity that is allocated within a custom price range. In earlier versions, liquidity was distributed uniformly along the price curve between 0 and infinity.

The previously uniform distribution allowed trading across the entire price interval $(0, \infty)$ without any loss of liquidity. However, in many pools, the majority of the liquidity was never used.

Consider stablecoin pairs, where the relative price of the two assets stays relatively constant. The liquidity outside the typical price range of a stablecoin pair is rarely touched. For example, the v2 DAI/USDC pair utilizes ~0.50% of the total available capital for trading between \$0.99 and \$1.01, the price range in which LPs would expect to see the most volume - and consequently earn the most fees.

With v3, liquidity providers may concentrate their capital to smaller price intervals than $(0, \infty)$. In a stablecoin/stablecoin pair, for example, an LP may choose to allocate capital solely to the 0.99 - 1.01 range. As a result, traders are offered deeper liquidity around the mid-price, and LPs earn more trading fees with their capital. We call liquidity concentrated to a finite interval a position. LPs may have many different positions per pool, creating individualized price curves that reflect the preferences of each LP.

Active Liquidity

As the price of an asset rises or falls, it may exit the price bounds that LPs have set in a position. When the price exits a position's interval, the position's liquidity is no longer active and no longer earns fees.

As price moves in one direction, LPs gain more of the one asset as swappers demand the other, until their entire liquidity consists of only one asset. (In v2, we don't typically see this behavior because LPs rarely reach the upper or lower bound of the price of two assets, i.e., 0 and ∞). If the price ever reenters the interval, the liquidity becomes active again, and in-range LPs begin earning fees once more.

Importantly, LPs are free to create as many positions as they see fit, each with its own price interval. Concentrated liquidity serves as a mechanism to let the market decide what a sensible distribution of liquidity is, as rational LPs are incentivized to concentrate their liquidity while ensuring that their liquidity remains active.

Ticks

To achieve concentrated liquidity, the once continuous spectrum of price space has been partitioned with ticks.

Ticks are the boundaries between discrete areas in price space. Ticks are spaced such that an increase or decrease of 1 tick represents a 0.01% increase or decrease in price at any point in price space.

Ticks function as boundaries for liquidity positions. When a position is created, the provider must choose the lower and upper tick that will represent their position's borders.

As the spot price changes during swapping, the pool contract will continuously exchange the outbound asset for the inbound, progressively using all the liquidity available within the current tick interval^[^1] until the next tick is reached. At this point, the contract switches to a new tick and activates any dormant liquidity within a position that has a boundary at the newly active tick.

While each pool has the same number of underlying ticks, in practice only a portion of them are able to serve as active ticks. Due to the nature of the v3 smart contracts, tick spacing is directly correlated to the swap fee. Lower fee tiers allow closer potentially active ticks, and higher fees

allow a relatively wider spacing of potential active ticks.

While inactive ticks have no impact on transaction cost during swaps, crossing an active tick does increase the cost of the transaction in which it is crossed, as the tick crossing will activate the liquidity within any new positions using the given tick as a border.

In areas where capital efficiency is paramount, such as stable coin pairs, narrower tick spacing increases the granularity of liquidity provisioning and will likely lower price impact when swapping - the result being significantly improved prices for stable coin swaps.

For more information on fee levels and their correlation to tick spacing, see the [whitepaper](#).

[^1]: Tick interval refers to the area of price space between two nearest active ticks

id: fees title: Fees sidebar_position: 2

Swap Fees

Swap fees are distributed pro-rata to all in-range[^1] liquidity at the time of the swap. If the spot price moves out of a position's range, the given liquidity is no longer active and does not generate any fees. If the spot price reverses and reenters the position's range, the position's liquidity becomes active again and will generate fees.

Swap fees are not automatically reinvested as they were in previous versions of Uniswap. Instead, they are collected separately from the pool and must be manually redeemed when the owner wishes to collect their fees.

Pool Fees Tiers

Uniswap v3 introduces multiple pools for each token pair, each with a different swapping fee. Liquidity providers may initially create pools at three fee levels: 0.05%, 0.30%, and 1%. More fee levels may be added by UNI governance, e.g. the 0.01% fee level added by [this](#) governance proposal in November 2021, as executed [here](#).

Breaking pairs into separate pools was previously untenable due to the issue of liquidity fragmentation. Any incentive alignments achieved by more fee optionality invariably resulted in a net loss to traders, due to lower pairwise liquidity and the resulting increase in price impact upon swapping.

The introduction of concentrated liquidity decouples total liquidity from price impact. With price impact concerns out of the way, breaking pairs into multiple pools becomes a feasible approach to improving the functionality of a pool for assets previously underserved by the 0.30% swap fee.

Finding The Right Pool Fee

We anticipate that certain types of assets will gravitate towards specific fee tiers, based on where the incentives for both swappers and liquidity providers come nearest to alignment.

We expect low volatility assets (stable coins) will likely congregate in the lowest fee tier, as the price risk for liquidity providers holding these assets is very low, and those swapping will be motivated to pursue an execution price closest to 1:1 as they can get.

Similarly, we anticipate more exotic assets, or those traded rarely, will naturally gravitate towards a higher fee - as liquidity providers will be motivated to offset the cost risk of holding these assets for the duration of their position.

Protocol Fees

Uniswap v3 has a protocol fee that can be turned on by UNI governance. Compared to v2, UNI governance has more flexibility in choosing the fraction of swap fees that go to the protocol. For details regarding the protocol fee, see the [whitepaper](#).

[^1]: In-range liquidity refers to the liquidity contained in any positions which span both sides of the spot price.

id: integration-issues title: Token Integration Issues sidebar_position: 6

Fee-on-transfer and rebasing tokens will not function correctly on v3.

Fee-on-transfer Tokens

Fee-on-transfer tokens will not function with our router contracts. As a workaround, the token creators may create a token wrapper or a customized router. We will not be making a router that supports fee-on-transfer tokens in the future.

Rebasing Tokens

Rebasing tokens will succeed in pool creation and swapping, but liquidity providers will bear the loss of a negative rebase when their position becomes active, with no way to recover the loss.

id: oracle title: Oracle sidebar_position: 3

:::note Unfamiliar with the concept of an oracle? Check out the Ethereum Foundation's [oracle overview](#) first. :::

All Uniswap v3 pools can serve as oracles, offering access to historical price and liquidity data. This capability unlocks a wide range of on-chain use cases.

Historical data is stored as an array of observations. At first, each pool tracks only a single observation, overwriting it as blocks elapse. This limits how far into the past users may access data. However, any party willing to pay the transaction fees may [increase the number of tracked observations](#) (up to a maximum of 65535), expanding the period of data availability to ~9 days or more.

Storing price and liquidity history directly in the pool contract substantially reduces the potential for logical errors on the part of the calling contract, and reduces integration costs by eliminating the need to store historical values. Additionally, the v3 oracle's considerable maximum length makes oracle price manipulation significantly more difficult, as the calling contract may cheaply construct a time-weighted average over any arbitrary range inside of (or fully encompassing) the length of the oracle array.

Observations

Observation s take the following form:

```
struct Observation {  
    // the block timestamp of the observation  
    uint32 blockTimestamp;  
    // the tick accumulator, i.e. tick * time elapsed since the pool was first initialized  
    int56 tickCumulative;  
    // the seconds per liquidity, i.e. seconds elapsed / max(1, liquidity) since the pool was first initialized  
    uint160 secondsPerLiquidityCumulativeX128;  
    // whether or not the observation is initialized  
    bool initialized;  
}
```

Observation s may be retrieved via the [observations](#) method on v3 pools. However, this is *not* the recommended way to consume oracle data. Instead, prefer [observe](#):

```
function observe(uint32[] calldata secondsAgo)  
external  
view  
returns (int56[] memory tickCumulatives, uint160[] memory secondsPerLiquidityCumulativeX128s);
```

Each time `observe` is called, the caller must specify an array containing any number of seconds ago, denoting the times to return observations from. Note that each of the given times must be more recent (or as old as) the oldest stored observation. Note: if the times don't correspond exactly to a block in which an observation was written, a counterfactual observation will be constructed, removing the need for the caller to interpolate manually. This is one of the primary reasons to use `observe over observations`.

Note that because the oracle is only updated at most once every block, calling `observe` with a `secondsAgo` value of 0 will return the most recently written observation, which can only be as recent as the beginning of the current block (or older).

Tick Accumulator

The tick accumulator stores the cumulative sum of the active tick at the time of the observation. The tick accumulator value increases monotonically and grows by the value of the current tick - per second.

To derive the arithmetic mean tick over an interval, the caller needs to retrieve two observations, one after the other, take the delta of the two values, and divide by the time elapsed between them. Calculating a TWAP from the tick accumulator is also covered in the [whitepaper](#). Note that using an arithmetic mean tick to derive a price corresponds to a *geometric* mean price.

See [OracleLibrary](#) for an example of how to use the tick accumulator.

Liquidity Accumulator

The liquidity accumulator stores the value of seconds / in-range liquidity at the time of the observation. The liquidity accumulator value increases monotonically and grows by the value of seconds / in-range liquidity - per second.

To derive the harmonic mean liquidity over an interval, the caller needs to retrieve two observations, one after the other, take the delta of the two values, then divide the time elapsed by this value. Calculating TWAL is addressed in finer detail in the [whitepaper](#).

:::note The in-range liquidity accumulator should be used with care. Because the current tick and the current in-range liquidity can be entirely uncorrelated, there are scenarios in which taking the arithmetic mean tick and the harmonic mean liquidity over the same interval in a pool can inaccurately characterize this pool relative to another. For example, if the current tick on pool A is 0 for 5 seconds, and 100 for 5 seconds, the tick accumulator will be 50. If over this same interval, the in-range liquidity was 5000 and 50, the harmonic mean liquidity will be ~99. Compare this to pool B (composed of the same assets) where the tick was 50 and the in-range liquidity was ~99 for 10 seconds. The accumulator values will be identical, but the underlying behavior is of course quite different. :::

Deriving Price From A Tick

When we use "active tick" or otherwise to refer to the current tick of a pool, we mean the lower tick boundary that is closest to the current price.

When a pool is created, each token is assigned to either `token0` or `token1` based on the contract address of the tokens in the pair. Whether or not a token is `token0` or `token1` is meaningless; it is only used to maintain a fixed assignment for the purpose of relative valuation and general logic in the pool contract.

Deriving an asset price from the current tick is achievable due to the fixed expression across the pool contract of `token0` in terms of `token1`.

An example of finding the price of WETH in a WETH / USDC pool, where WETH is `token0` and USDC is `token1`:

You have an oracle reading that shows a return of `tickCumulative as [70_000 , 1_070_000]`, with an elapsed time between the observations of 10 seconds.

We can derive the average tick over this interval by taking the difference in accumulator values ($1_{070_{000}} - 70_{000} = 1_{000_{000}}$), and dividing by the time elapsed ($1_{000_{000}} / 10 = 100_{000}$).

With a tick reading of `100_000`, we can find the value of `token1` (USDC) in terms of `token0` (WETH) by using the current tick as `i` in the formula `p(i) = 1.0001**i` (see 6.1 in the [whitepaper](#)).

$1.0001^{100_{000}} \approx 22015.5 \text{ USDC / WETH}$

Ticks are signed integers and can be expressed as a negative number, so for any circumstances where `token0` is of a lower value than `token1`, a negative tick value will be returned by `tickCumulative` and a relative value of `< 0` will be returned by a calculation of `token0` in terms of `token1`.

Oracles Integrations on Layer 2 Rollups

Optimism

On Optimism, every transaction is confirmed as an individual block. The `block.timestamp` of these blocks, however, reflect the `block.timestamp` of the last L1 block ingested by the Sequencer. For this reason, Uniswap pools on Optimism are not suitable for providing oracle prices, as this high-latency `block.timestamp` update process makes the oracle much less costly to manipulate. In the future, it's possible that the Optimism `block.timestamp` will have much higher granularity (with a small trust assumption in the Sequencer), or that forced inclusion transactions will improve oracle security. For more information on these potential upcoming changes, please see the [Optimistic Specs repo](#). For the time being, usage of the oracle feature on Optimism should be avoided.

id: range-orders title: Range Orders sidebar_position: 4

Customizable liquidity positions, along with single-sided asset provisioning, allow for a new style of swapping with automated market makers: the range order.

In typical order book markets, anyone can easily set a limit order: to buy or sell an asset at a specific predetermined price, allowing the order to be filled at an indeterminate time in the future.

With Uniswap V3, one can approximate a limit order by providing a single asset as liquidity within a specific range. Like traditional limit orders, range orders may be set with the expectation they will execute at some point in the future, with the target asset available for withdrawal after the spot price has crossed the full range of the order.

Unlike some markets where limit orders may incur fees, the range order maker generates fees while the order is filled. This is due to the range order technically being a form of liquidity provisioning rather than a typical swap.

Possibilities of Range orders

The nature of AMM design makes some styles of limit orders possible, while others cannot be replicated. The following are four examples of range orders and their traditional counterparts; the first two are possible, the second two are not.

One important distinction: range orders, unlike traditional limit orders, will be **unfilled** if the spot price crosses the given range and then reverses to recross in the opposite direction before the target asset is withdrawn. While you will be earning LP fees during this time, if the goal is to exit fully in the desired destination asset, you will need to keep an eye on the order and either manually remove your liquidity when the order has been filled or use a third party position manager service to withdraw on your behalf.

Take-Profit Orders

The current price of a DAI / ETH pool is 1,500 DAI / ETH. You would like to sell your ETH for DAI when the price of ETH reaches 1,600 DAI / ETH. This is possible, as the price space above the spot price is denominated in the higher valued asset, ETH. You can provide ETH at a price of 1,600 DAI / ETH and have the order filled when the spot price crosses your position.

Buy Limit Orders

The Current price of a DAI / ETH pool is 1,500 DAI / ETH. You expect that ETH will rebound after it drops to 1,000 at the next market downturn, so you would like to place a range order swapping DAI to ETH at the price of 1,000 DAI / ETH. This is possible, as the price space below the spot price is denominated in the lower-priced asset, DAI. You can provide DAI at the price of 1,000 DAI / ETH, which will be swapped for ETH when the spot price of ETH drops past 1,000 DAI / ETH.

As the above examples show, in Uniswap V3, the two paired assets in a given pool are separated above and below the spot price, with the higher price asset available above the spot price and the lower-priced asset below.

The following examples show limit order styles that are unable to be replicated due to the separation of assets in price space.

Buy Stop Orders

The current price of a DAI / ETH pool is 1,500 DAI / ETH. You expect the price of ETH to rocket up to 3,000 once it hits 2,000 DAI / ETH. So you would like to place a range order from DAI to ETH at a price of 2,000 DAI / ETH. This is not possible as the price space above 1,500 DAI / ETH is denominated in ETH - and thus, you cannot provide the DAI necessary at your desired price to be swapped into ETH.

Stop-Loss Orders

The current price of a DAI / ETH pool is 1,500 DAI / ETH. You expect once the price of ETH drops to below 1,000, it will tank to 200. So you would like to place a range order from ETH to DAI at a price of 1,000. This is not possible as the price space below the spot price is denominated in DAI, and so you cannot allocate the ETH necessary at 1,000 to be swapped into DAI.

Fees

The fees due to your liquidity position will be denominated in both tokens of the given pair. In any of the above examples, after swapping ETH for DAI, or DAI for ETH, a small amount of both ETH and DAI will be due to your account as liquidity provisioning rewards.

Approaches to concentration when setting range orders are up to the user. Selecting a wider range may generate more fees if there is price churn within your range, at the cost of increasing the risk of having your order unfilled if the spot price reverses before completing your full range.

id: swaps title: Swaps sidebar_position: 5

Introduction

Swaps are the most common way of interacting with the Uniswap protocol. For end-users, swapping is straightforward: a user selects an ERC-20 token that they own and a token they would like to trade it for. Executing a swap sells the currently owned tokens for the proportional^[^1] amount of the tokens desired, minus the swap fee, which is awarded to liquidity providers^[^2]. Swapping with the Uniswap protocol is a permissionless process.

note: Using web interfaces (websites) to swap via the Uniswap protocol can introduce additional permission structures, and may result in different execution behavior compared to using the Uniswap protocol directly. To learn more about the differences between the protocol and a web interface, see [What is Uniswap](#).

Swaps using the Uniswap protocol are different from traditional order book trades in that they are not executed against discrete orders on a first-in-first-out basis — rather, swaps execute against a passive pool of liquidity, with liquidity providers earning fees proportional to their capital committed

Price Impact

In a traditional order-book market, a sizeable market-buy order may deplete the available liquidity of a prior limit-sell and continue to execute against a subsequent limit-sell order at a higher price. The result is the final execution price of the order is somewhere in between the two limit-sell prices against which the order was filled.

Price impact affects the execution price of a swap similarly but is a result of a different dynamic. When using an automated market maker, the relative value of one asset in terms of the other continuously shifts during the execution of a swap, leaving the final execution price somewhere between where the relative price started - and ended.

This dynamic affects every swap using the Uniswap protocol, as it is an inextricable part of AMM design.

As the amount of liquidity available at different price points can vary, the price impact for a given swap size will change relative to the amount of liquidity available at any given point in price space. The greater the liquidity available at a given price, the lower the price impact for a given swap size. The lesser the liquidity available, the higher the price impact.

Approximate^[^3] price impact is anticipated in real-time via the Uniswap interface, and warnings appear if unusually high price impact will occur during a swap. Anyone executing a swap will have the ability to assess the circumstances of price impact when needed.

Slippage

The other relevant detail to consider when approaching swaps with the Uniswap protocol is slippage. Slippage is the term we use to describe alterations to a given price that could occur while a submitted transaction is pending.

When transactions are submitted to Ethereum, their order of execution is established by the amount of "gas" offered as a fee for executing each transaction. The higher the fee offered, the faster the transaction is executed. The transactions with a lower gas fee will remain pending for an indeterminate amount of time. During this time, the price environment in which the transaction will eventually be executed will change, as other swaps will be taking place.

Slippage tolerances establish a margin of change acceptable to the user beyond price impact. As long as the execution price is within the slippage range, e.g., $\pm 1\%$, the transaction will be executed. If the execution price ends up outside of the accepted slippage range, the transaction will fail, and the swap will not occur.

A comparable situation in a traditional market would be a market-buy order executed after a delay. One can know the expected price of a market-buy order when submitted, but much can change in the time between submission and execution.

Safety Checks

Price impact and slippage can both change while a transaction is pending, which is why we have built numerous safety checks into the Uniswap protocol to protect end-users from drastic changes in the execution environment of their swap. Some of the most commonly encountered safety checks:

- **Expired** : A transaction error that occurs if a swap is pending longer than a predetermined deadline. The deadline is a point in time after which the swap will be canceled to protect against unusually long pending periods and the changes in price that typically accompany the passage of time.
- **INSUFFICIENT_OUTPUT_AMOUNT** : When a user submits a swap, the Uniswap interface will send an estimate of how much of the purchased token the user should expect to receive. If the anticipated output amount of a swap does not match the estimate within a certain margin of error (the slippage tolerance), the swap will be canceled. This attempts to protect the user from any drastic and unfavorable price changes while their transaction is pending.

[^1]: Proportional in this instance takes into account many factors, including the relative price of one token in terms of the other, slippage, price impact, and other factors related to the open and adversarial nature of Ethereum. [^2]: For information about liquidity provision, see the liquidity user guide [^3]: The Uniswap interface informs the user about the circumstances of their swap, but it is not guaranteed.

id: research title: Research sidebar_position: 4

The automated market maker is a new concept, and as such, new research comes out frequently. We've selected some of the most thoughtful here.

Uniswap's Financial Alchemy

Authors: Dave White, Martin Tassy, Charlie Noyes, and Dan Robinson

An automated market maker is a type of decentralized exchange that lets customers trade between on-chain assets like USDC and ETH. Uniswap is the most popular AMM on Ethereum. Like most AMMs, Uniswap facilitates trading between a particular pair of assets by holding reserves of both assets. It sets the trading price between them based on the size of its reserves in such a way that prices will stay in line with the broader market. Anybody who would like to can join the "pool" for a particular pair and become a liquidity provider, or LP, so-called because they provide liquid assets for others to trade against. LPs contribute assets to both reserves simultaneously, taking on some of the risk of trading in exchange for a share of the returns.

- [Uniswap's Financial Alchemy](#)

An analysis of Uniswap markets

Authors: Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, Tarun Chitra

Uniswap---and other constant product markets---appear to work well in practice despite their simplicity. In this paper, we give a simple formal analysis of constant product markets and their generalizations, showing that, under some common conditions, these markets must closely track the reference market price. We also show that Uniswap satisfies many other desirable properties and numerically demonstrate, via a large-scale agent-based simulation, that Uniswap is stable under a wide range of market conditions.

- [An analysis of Uniswap markets](#)

Improved Price Oracles: Constant Function Market Makers

Authors: Guillermo Angeris, Tarun Chitra

Automated market makers, first popularized by Hanson's logarithmic market scoring rule (or LMSR) for prediction markets, have become important building blocks, called 'primitives,' for decentralized finance. A particularly useful primitive is the ability to measure the price of an asset, a problem often known as the pricing oracle problem. In this paper, we focus on the analysis of a very large class of automated market makers, called constant function market makers (or CFMMs) which includes existing popular market makers such as Uniswap, Balancer, and Curve, whose yearly transaction volume totals to billions of dollars. We give sufficient conditions such that, under fairly general assumptions, agents who interact with these constant function market makers are incentivized to correctly report the price of an asset and that they can do so in a computationally efficient way. We also derive several other useful properties that were previously not known. These include lower bounds on the total value of assets held by CFMMs and lower bounds guaranteeing that no agent can, by any set of trades, drain the reserves of assets held by a given CFMM.

- [Improved Price Oracles: Constant Function Market Makers](#)

Pintail research

Published [medium](#) articles by Pintail.

- [Understanding Uniswap Returns](#)
- [Uniswap: A Good Deal for Liquidity Providers?](#)

Liquidity Provider Returns in Geometric Mean Markets

Authors: Alex Evans

Geometric mean market makers (G3Ms), such as Uniswap and Balancer, comprise a popular class of automated market makers (AMMs) defined by the following rule: the reserves of the AMM before and after each trade must have the same (weighted) geometric mean. This paper extends several results known for constant-weight G3Ms to the general case of G3Ms with time-varying and potentially stochastic weights. These results include the returns and no-arbitrage prices of liquidity pool (LP) shares that investors receive for supplying liquidity to G3Ms. Using these expressions, we show how to create G3Ms whose LP shares replicate the payoffs of financial derivatives. The resulting hedges are model-independent and exact for derivative contracts whose payoff functions satisfy an elasticity constraint. These strategies allow LP shares to replicate various trading strategies and financial contracts, including standard options. G3Ms are thus shown to be capable of recreating a variety of active trading strategies through passive positions in LP shares.

- [Liquidity Provider Returns in Geometric Mean Markets](#)

The Replicating Portfolio of a Constant Product Market

Authors: Joseph Clark

We derive the replicating portfolio of a constant product market. This is structurally short volatility (selling options) which explains why positive transaction costs are needed to induce liquidity providers to participate. Where futures and options markets do not exist, this payoff can be used to create them.

- https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3550601

id: resources title: Resources sidebar_position: 5

This page will be periodically updated with helpful resources for calculations and software design as they are made available.

- [Uniswap V2 Visualization](#)
- [Uniswap V3 Visualization](#)
- [Uniswap V3 \(Multiple Positions\) Visualization](#)
- [V2 / V3 Calculations](#)

id: uniswap-protocol title: The Uniswap Protocol sidebar_position: 2

Introduction

The Uniswap protocol is a peer-to-peer^[^1] system designed for exchanging cryptocurrencies ([ERC-20 Tokens](#)) on the [Ethereum](#) blockchain. The protocol is implemented as a set of persistent, non-upgradable smart contracts; designed to prioritize censorship resistance, security, self-custody, and to function without any trusted intermediaries who may selectively restrict access.

There are currently three versions of the Uniswap protocol. V1 and V2 are open source and licensed under GPL. V3 is open source with slight modifications, which are viewable [here](#). Each version of Uniswap, once deployed, will function in perpetuity, with 100% uptime, provided the continued existence of the Ethereum blockchain.

How does the Uniswap protocol compare to a typical market?

To understand how the Uniswap protocol differs from a traditional exchange, it is helpful to first look at two subjects: how the Automated Market Maker design deviates from traditional central limit order book-based exchanges, and how permissionless systems depart from conventional permissioned systems.

Order Book VS AMM

Most publicly accessible markets use a central limit [order book](#) style of exchange, where buyers and sellers create orders organized by price level that are progressively filled as demand shifts. Anyone who has traded stocks through brokerage firms will be familiar with an order book system.

The Uniswap protocol takes a different approach, using an Automated Market Maker (AMM), sometimes referred to as a Constant Function Market Maker, in place of an order book.

At a very high level, an AMM replaces the buy and sell orders in an order book market with a liquidity pool of two assets, both valued relative to each other. As one asset is traded for the other, the relative prices of the two assets shift, and a new market rate for both is determined. In this dynamic, a buyer or seller trades directly with the pool, rather than with specific orders left by other parties. The advantages and disadvantages of Automated Market Makers versus their traditional order book counterparts are under active research by a growing number of parties. We have collected some notable examples on our [research page](#).

Permissionless Systems

The second departure from traditional markets is the permissionless and immutable design of the Uniswap protocol. These design decisions were inspired by Ethereum's core tenets, and our commitment to the ideals of permissionless access and immutability as indispensable components of a future in which anyone in the world can access financial services without fear of discrimination or counter-party risk.

Permissionless design means that the protocol's services are entirely open for public use, with no ability to selectively restrict who can or cannot use them. Anyone can swap, provide liquidity, or create new markets at will. This is a departure from traditional financial services, which typically restrict access based on geography, wealth status, and age.

The protocol is also immutable, in other words not upgradeable. No party is able to pause the contracts, reverse trade execution, or otherwise change the behavior of the protocol in any way. It is worth noting that Uniswap Governance has the right (but no obligation) to divert a percentage of swap fees on any pool to a specified address. However, this capability is known to all participants in advance, and to prevent abuse, the percentage is constrained between 10% and 25%.

Where can I find more information

For research into the economics of AMMs, game theory, or optimization research, check out our [research](#) page.

For new features implemented in V3 that expand and refine the AMM design, see the [V3 Concepts](#) page.

[^1]: Ethereum protocols are sometimes referred to as peer-to-contract systems as well. These are similar to a peer-to-peer systems, but with immutable, persistent programs known as smart contracts taking the place of a peer.

id: overview title: Uniswap Overview sidebar_position: 1

Protocol, Interface, Labs

To begin, we should make clear the distinctions between the different areas of "Uniswap", some of which may confuse new users.

- **Uniswap Labs:** The company which developed the Uniswap protocol, along with the web interface.
- **The Uniswap Protocol:** A suite of persistent, non-upgradable smart contracts that together create an automated market maker, a protocol that facilitates peer-to-peer market making and swapping of ERC-20 tokens on the Ethereum blockchain.
- **The Uniswap Interface:** A web interface that allows for easy interaction with the Uniswap protocol. The interface is only one of many ways one may interact with the Uniswap protocol.
- **Uniswap Governance:** A governance system for governing the Uniswap Protocol, enabled by the UNI token.

id: overview title: Overview sidebar_position: 1

[Permit2](#) is a unification of 2 contracts, [SignatureTransfer](#) and [AllowanceTransfer](#). The [SignatureTransfer](#) contract handles all signature-based transfers, meaning that an allowance on the token is bypassed and permissions to the spender only last for the duration of the

transaction that the one-time signature is spent. The `AllowanceTransfer` contract handles setting allowances on tokens, giving permissions to spenders on a specified amount for a specified duration of time. Any transfers that then happen through the `AllowanceTransfer` contract will only succeed if the proper permissions have been set.

Resources

A great [explanation](#) of the Permit2 contract and example usage.

Approving Permit2

Before integrating contracts can request users' tokens through Permit2, users must approve the Permit2 contract through the specific token contract by calling something like:

```
USDC.approve(permit2Address, totalAmount);
```

To get the maximal benefits from Permit2, users should do a max approval on the contract where:

```
totalAmount = type(uint256).max;
```

id: allowance-transfer title: AllowanceTransfer sidebar_position: 2

[Source Code](#)

Overview

The main entry points on this contract are:

- `approve`
 - Use approve when you do not want to set token permissions through signature validation.
- `permit`
 - Use permit when you do want to set token permissions through signature validation.
- `transferFrom`
 - Use transferFrom when you want to transfer a token and have the necessary permissions to do so.

Functions

approve

Function Signature

```
function approve(address token, address spender, uint160 amount, uint48 expiration) external
```

Parameters

- `token` - the token address to approve
- `spender` - the spender address to approve
- `amount` - the approved amount of the token, `type(uint160).max` is treated as an unlimited allowance
- `expiration` - the timestamp at which the approval is no longer valid, passing in `0` will expire the permissions at `block.timestamp`

Single permit

Function Signature

```
function permit(address owner, PermitSingle memory permitSingle, bytes calldata signature) external;
```

Parameters

- `owner` - the address of the token's owner
- `permitSingle` - constructed with the following:

```
struct PermitSingle {  
    // the permit data for a single token allowance  
    PermitDetails details;  
    // address permissioned on the allowed tokens  
    address spender;  
    // deadline on the permit signature
```

```

        uint256 sigDeadline;
    }

struct PermitDetails {
    // ERC20 token address
    address token;
    // the maximum amount allowed to spend
    uint160 amount;
    // timestamp at which a spender's token allowances become invalid
    uint48 expiration;
    // an incrementing value indexed per owner,token, and spender for each signature
    uint48 nonce;
}

```

- **signature** - the signature over the permit data. Supports EOA signatures, compact signatures defined by [EIP-2098](#), and contract signatures defined by [EIP-1271](#)

Batched permit

Function Signature

```
function permit(address owner, PermitBatch memory permitBatch, bytes calldata signature) external;
```

Parameters

- **owner** - the address of the token's owner
- **permitBatch** - constructed with the following:

```

struct PermitBatch {
    // the permit data for multiple token allowances
    PermitDetails[] details;
    // address permissioned on the allowed tokens
    address spender;
    // deadline on the permit signature
    uint256 sigDeadline;
}

struct PermitDetails {
    // ERC20 token address
    address token;
    // the maximum amount allowed to spend
    uint160 amount;
    // timestamp at which a spender's token allowances become invalid
    uint48 expiration;
    // an incrementing value indexed per owner,token, and spender for each signature
    uint48 nonce;
}

```

- **signature** - the signature over the permit data. Supports EOA signatures, compact signatures defined by [EIP-2098](#), and contract signatures defined by [EIP-1271](#)

Single transferFrom

Function Signature

```
function transferFrom(address from, address to, uint160 amount, address token) external;
```

Parameters

- **from** - the address to transfer the token from
- **to** - the address of the recipient
- **amount** - the amount of the token to transfer, the maximum amount is `type(uint160).max`
- **token** - the address of the token to be transferred

Batched transferFrom

Function Signature

```
function transferFrom(AllowanceTransferDetails[] calldata transferDetails) external;
```

Parameters

- transferDetails - constructed with the following

```
struct AllowanceTransferDetails {
    // the owner of the token
    address from;
    // the recipient of the token
    address to;
    // the amount of the token
    uint160 amount;
    // the token to be transferred
    address token;
}
```

Nonce Schema

The nonces used to protect against replay attacks of signatures are similar to standard incrementing nonces. However, we pack nonces with an allowed amount, and an allowed duration. This means that nonces are incremented *per owner*, *per token*, and *per spender*. Which further implies that you could sign two different permits at the same time with the same nonces and they won't cancel each other out so long as the token or spender differ.

The mapping nonces are packed in is defined as follows:

```
mapping(address => mapping(address => mapping(address => PackedAllowance))) public allowance;
```

and indexed as follows:

```
PackedAllowance allowanceInformation = allowance[ownerAddress][tokenAddress][spenderAddress];
uint48 nonce = allowanceInformation.nonce;
```

Security Considerations

Similar to the security considerations outlined in `SignatureTransfer`, integrating contracts need to perform valid safety checks on the caller and pass in correct addresses for the `from` argument in any transfer calls.

All amounts on the `AllowanceTransfer` contract are of type `uint160` so make sure integrating contracts safely downcast if they have to. See how `Permit2Lib` downcasts safely.---
id: signature-transfer title: SignatureTransfer sidebar_position: 1

[Source Code](#)

Overview

The main entry points on this contract are:

- `permitTransferFrom`
 - Use `permitTransferFrom` when you want to transfer a token from an owner through signature validation.
- `permitWitnessTransferFrom`
 - Use `permitWitnessTransferFrom` when you want to transfer a token from an owner through signature validation, but you would also like to validate other data. Any other data you wish to be validated can be passed through with the `witness` param.

Each of these functions is overloaded with a batched version that allows users to transfer multiple tokens with 1 transaction.

Functions

Single `permitTransferFrom`

Use the `permitTransferFrom` to transfer just one token.

Function signature

```
function permitTransferFrom(
    PermitTransferFrom memory permit,
    SignatureTransferDetails calldata transferDetails,
    address owner,
```

```
    bytes calldata signature  
) external
```

Parameters

- `permit` - Construct `PermitTransferFrom` struct with the following:

```
struct PermitTransferFrom {  
    TokenPermissions permitted;  
    // a unique value for every token owner's signature to prevent signature replays  
    uint256 nonce;  
    // deadline on the permit signature  
    uint256 deadline;  
}  
  
struct TokenPermissions {  
    // ERC20 token address  
    address token;  
    // the maximum amount that can be spent  
    uint256 amount;  
}
```

- `transferDetails` - information about recipient and amount

```
struct SignatureTransferDetails {  
    // recipient address  
    address to;  
    // spender requested amount  
    uint256 requestedAmount;  
}
```

- `owner` - the signer of the permit message and owner of the tokens
- `signature` - the signature over the permit data. Supports EOA signatures, compact signatures defined by [EIP-2098](#), and contract signatures defined by [EIP-1271](#)

Batched `permitTransferFrom`

Use `permitTransferFrom` with the batched parameters when you want to transfer multiple tokens from an owner.

Function Signature

```
function permitTransferFrom(  
    PermitBatchTransferFrom memory permit,  
    SignatureTransferDetails[] calldata transferDetails,  
    address owner,  
    bytes calldata signature  
) external
```

Parameters

- `permit` - Construct `PermitBatchTransferFrom` with the following:

```
struct PermitBatchTransferFrom {  
    // the tokens and corresponding amounts permitted for a transfer  
    TokenPermissions[] permitted;  
    // a unique value for every token owner's signature to prevent signature replays  
    uint256 nonce;  
    // deadline on the permit signature  
    uint256 deadline;  
}  
  
struct TokenPermissions {  
    // ERC20 token address  
    address token;  
    // the maximum amount that can be spent  
    uint256 amount;  
}
```

- `transferDetails` - parameterized by the spender with information about the token transfer.

- o The length of the `SignatureTransferDetails` array must equal the length of the `TokenPermissions` array passed in with `PermitBatchTransferFrom` struct. The token to be transferred specified in the `TokenPermissions` array should match the index of the `SignatureTransferDetails` array.
- o Note that if a spender is permitted to a token but does not need to transfer that token, they can specify that the `requestedAmount` is 0 so that the transfer is skipped.
- owner - the signer of the permit message and owner of the tokens

```
struct SignatureTransferDetails {
    // recipient address
    address to;
    // spender requested amount
    uint256 requestedAmount;
}
```

- signature - the signature over the permit data. Supports EOA signatures, compact signatures defined by [EIP-2098](#), and contract signatures defined by [EIP-1271](#)

Single `permitWitnessTransferFrom`

Function Signature

```
function permitWitnessTransferFrom(
    PermitTransferFrom memory permit,
    SignatureTransferDetails calldata transferDetails,
    address owner,
    bytes32 witness,
    string calldata witnessTypeString,
    bytes calldata signature
) external
```

Parameters

- permit - constructed with the same type as defined above in the single `permitTransferFrom` case
- transferDetails constructed with same type as defined above in the single `permitTransferFrom` case
- owner - the signer of the permit message and owner of the tokens
- witness - arbitrary data passed through that was signed by the user. Is used to reconstruct the signature. Pass through this data if you want the permit signature recovery also to validate other data.
- witnessTypeString - a string that defines the typed data that the witness was hashed from. It must also include the `TokenPermissions` struct and comply with [EIP-712](#) struct ordering. See an example below.
- signature - the signature over the permit data. Supports EOA signatures, compact signatures defined by [EIP-2098](#), and contract signatures defined by [EIP-1271](#)

Batch `permitWitnessTransferFrom`

Function Signature

```
function permitWitnessTransferFrom(
    PermitBatchTransferFrom memory permit,
    SignatureTransferDetails[] calldata transferDetails,
    address owner,
    bytes32 witness,
    string calldata witnessTypeString,
    bytes calldata signature
) external
```

Parameters

- permit - constructed with the same type in the batched case of `permitTransferFrom`
- transferDetails - constructed with the same type in the batched case of `permitTransferFrom`
- owner - the signer of the permit message and owner of the tokens
- witness - arbitrary data passed through that was signed by the user. Is used to reconstruct the signature. Pass through this data if you want the permit signature recovery to also validate other data.
- witnessTypeString - a string that defines the typed data that the witness was hashed from. It must also include the `TokenPermissions` struct and comply with [EIP-712](#) struct ordering. See an example below.
- signature - the signature over the permit data. Supports EOA signatures, compact signatures defined by [EIP-2098](#), and contract signatures defined by [EIP-1271](#)

Example `permitWitnessTransferFrom` parameters

If an integrating contract would also like the signer to verify information about a trade, an integrating contract may ask the signer to also sign an `ExampleTrade` object that we define below:

```
struct ExampleTrade {  
    address exampleTokenAddress;  
    uint256 exampleMinimumAmountOut;  
}
```

Following EIP-721, the typehash for the data would be defined by:

```
bytes32 _EXAMPLE_TRADE_TYPEHASH = keccak256('ExampleTrade(address exampleTokenAddress,uint256  
exampleMinimumAmountOut)');
```

The `witness` that should be passed along with the permit message should be:

```
bytes32 witness = keccak256(  
    abi.encode(_EXAMPLE_TRADE_TYPEHASH, exampleTrade.exampleTokenAddress,  
    exampleTrade.exampleMinimumAmountOut));
```

And the `witnessTypeString` to be passed in should be:

```
string constant witnessTypeString = "ExampleTrade witness)ExampleTrade(address exampleTokenAddress,uint256  
exampleMinimumAmountOut)TokenPermissions(address token,uint256 amount)"
```

It's important to note that when hashing multiple typed structs, the ordering of the structs in the type string matters. Referencing EIP-721:

If the struct type references other struct types (and these in turn reference even more struct types), then the set of referenced struct types is collected, sorted by name and appended to the encoding. An example encoding is Transaction(Person from,Person to,Asset tx)Asset(address token,uint256 amount)Person(address wallet,string name)

Nonce Schema

Instead of using incrementing nonces, we introduce non-monotonic, or unordered nonces with a `nonceBitmap`.

The `nonceBitmap` maps an owner's address to a `uint248` value, which we will call `wordPos` which is the index of the desired bitmap. There are **2248 possible indices and this 2248** possible bitmaps where each bitmap holds 256 bits. A bit must be flipped on to prevent replays of users' signatures. Bits that are dirtied may not be used again.

```
// nonceBitmap[ownerAddress][wordPosition] retrieves a uint256 bitmap  
mapping(address => mapping(uint248 => uint256)) public nonceBitmap;
```

Users will sign a `uint256` `nonce` value where the first 248 bits correspond to the word position of the bitmap to dirty and the last 8 bits correspond to the actual bit position being flipped on.

```
uint248 wordPos = uint248(nonce >> 8);  
uint8 bitPos = uint8(nonce);
```

```
uint256 bitmap = nonceBitmap[wordPos][bitPos]
```

Security Considerations

An integrating contract must check that tokens are released by a triggering call from the signer, or that the signer meant for their signature to be released by someone else.

💡 Consider this scenario:

A signer called Bob signs a permit to transfer 100 USDC with a router contract as the permissioned spender. The router contract never checks who the caller is but spends any permit messages on the Permit2 contract. An attacker Eve can steal Bob's signature, pass it through to the router with herself as the recipient, and transfer Bob's tokens to herself.

Universal Router protects against this by checking that the `msg.sender` from inside the routing contract is the supposed spender by passing `msg.sender` in as the `owner` param in any permit

calls and by passing in `msg.sender` as the `from` param in any transfer calls.--- id: overview

title: Overview sidebar_position: 1

The `UniversalRouter` is an ETH, ERC20, and NFT swap router, that can aggregate trades across protocols to give users access highly-flexible and personalised transactions. The contract is unowned, and is not upgradeable.

The flexible command style allows us to provide users with:

- Splitting and interleaving of Uniswap trades
- Purchases of NFTs across 8 marketplaces
- Partial fills of trades
- Wrapping and Unwrapping of ETH
- Time-bound, signature controlled token approvals using [Permit2](#)

Transactions are encoded using a string of commands, allowing users to have maximum flexibility over what they want to perform. With all of these features available in a single transaction, the possibilities available to users are endless.

Note: The `UniversalRouter` uses [Permit2](#) to remove the need for token approvals being provided directly to the `UniversalRouter`. The [Permit2 documentation](#) can be found [here](#).

id: technical-reference title: Technical Reference sidebar_position: 1

Functions

Transactions to the `UniversalRouter` all go through the `UniversalRouter.execute` functions:

- `execute(bytes calldata commands, bytes[] calldata inputs, uint256 deadline)`
- `execute(bytes calldata commands, bytes[] calldata inputs)`

The first of these functions adds the functionality to allow transactions to have a transaction deadline. If the `block.timestamp` is after the `deadline` provided the transaction will revert. After that check, the 2 functions otherwise execute identically.

The `execute` functions work like a simplified VM - they take in a list of commands, and a list of inputs for the commands and execute them in the order specified.

Command Structure

The first parameter for the function (`bytes calldata commands`) is a list of commands for the contract to execute, in the order they should be executed. Each command is encoded in 1 byte, containing the following split of 8 bits:

0	12	3 4 5 6 7
f	r	command

f

A single bit flag, that signals whether or not the command should be allowed to revert without the whole transaction failing.

- If `f` is `0` aka `false` and the command reverts, then the entire transaction will revert and none of the commands will be executed.
- If `f` is `1` aka `true` and the command reverts, then the transaction will continue, allowing us to achieve partial fills. If using this flag, be careful to include further commands that will remove any funds that could be left unused in the `UniversalRouter` contract.

r

2 unused bytes, reserved for future use. Leaving these 2 bits as `0` will save gas, but any value passed into the contract will be ignored. Later versions of the `UniversalRouter` will likely expand the 5 bits used for `command` to use at least 1 of these bits.

command

A 5 bit unique identifier for the command that should be carried out. The values of these commands can be found within [Commands.sol](#), or can be viewed in the table below.

The command types that are not defined do not have an assigned command at this moment in time. Providing one of these identifiers will cause the transaction to revert with `InvalidCommandType`.

A complete list of commands can be found in the table below:

Command	Value
0x00	V3_SWAP_EXACT_IN

0x01	<u>V3_SWAP_EXACT_OUT</u>
0x02	<u>PERMIT2_TRANSFER_FROM</u>
0x03	<u>PERMIT2_PERMIT_BATCH</u>
0x04	<u>SWEEP</u>
0x05	<u>TRANSFER</u>
0x06	<u>PAY_PORTION</u>
0x07	
0x08	<u>V2_SWAP_EXACT_IN</u>
0x09	<u>V2_SWAP_EXACT_OUT</u>
0x0a	<u>PERMIT2_PERMIT</u>
0x0b	<u>WRAP_ETH</u>
0x0c	<u>UNWRAP_WETH</u>
0x0d	<u>PERMIT2_TRANSFER_FROM_BATCH</u>
0x0e	
0x0f	
0x10	<u>SEAPORT</u>
0x11	<u>LOOKS_RARE_721</u>
0x12	<u>NFTX</u>
0x13	<u>CRYPTOPUNKS</u>
0x14	<u>LOOKS_RARE_1155</u>
0x15	<u>OWNER_CHECK_721</u>
0x16	<u>OWNER_CHECK_1155</u>
0x17	<u>SWEEPERC721</u>
0x18	<u>X2Y2_721</u>
0x19	<u>SUDOSWAP</u>
0x1a	<u>NFT20</u>
0x1b	<u>X2Y2_1155</u>
0x1c	<u>FOUNDATION</u>
0x1d	<u>SWEEPERC1155</u>
0x1e	
0x1f	

Command Inputs

The second parameter for the function is an array of bytes strings. Each element in the array is the abi-encoded input that will be used for the respective command.

`commands[i]` is the command that will use `inputs[i]` as its encoded input parameters.

The router uses the command type to know how to decode the encoded input parameters - depending on the command chosen, the required inputs is different.

The input parameters required for each command are outlined below:

V3_SWAP_EXACT_IN

- `address` The recipient of the output of the trade
- `uint256` The amount of input tokens for the trade

- `uint256` The minimum amount of output tokens the user wants
- `bytes` The UniswapV3 encoded path to trade along
- `bool` A flag for whether the input tokens should come from the `msg.sender` (through Permit2) or whether the funds are already in the UniversalRouter

V3_SWAP_EXACT_OUT

- `address` The recipient of the output of the trade
- `uint256` The amount of output tokens to receive
- `uint256` The maximum number of input tokens that should be spent
- `bytes` The UniswapV3 encoded path to trade along
- `bool` A flag for whether the input tokens should come from the `msg.sender` (through Permit2) or whether the funds are already in the UniversalRouter

PERMIT2_TRANSFER_FROM

- `address` The token to fetch from Permit2
- `address` The recipient of the tokens fetched
- `uint256` The amount of token to fetch

The individual that the tokens are fetched from is always the `msg.sender` of the transaction

PERMIT2_PERMIT_BATCH

- `IAllowanceTransfer.PermitsBatch` A `PermitBatch` struct outlining all of the Permit2 permits to execute.
- `bytes` The signature to provide to Permit2

The individual that signed the permits must be the `msg.sender` of the transaction

SWEEP

- `address` The ERC20 token to sweep (or Constants.ETH for ETH)
- `address` The recipient of the sweep
- `uint256` The minimum required tokens to receive from the sweep

TRANSFER

- `address` The ERC20 token to transfer (or Constants.ETH for ETH)
- `address` The recipient of the transfer
- `uint256` The amount to transfer

PAY_PORTION

- `address` The ERC20 token to transfer (or Constants.ETH for ETH)
- `address` The recipient of the transfer
- `uint256` In basis points, the percentage of the contract's balance to transfer

V2_SWAP_EXACT_IN

- `address` The recipient of the output of the trade
- `uint256` The amount of input tokens for the trade
- `uint256` The minimum amount of output tokens the user wants
- `address[]` The UniswapV2 token path to trade along
- `bool` A flag for whether the input tokens should come from the `msg.sender` (through Permit2) or whether the funds are already in the UniversalRouter

V2_SWAP_EXACT_OUT

- `address` The recipient of the output of the trade
- `uint256` The amount of output tokens to receive
- `uint256` The maximum number of input tokens that should be spent
- `address[]` The UniswapV2 token path to trade along
- `bool` A flag for whether the input tokens should come from the `msg.sender` (through Permit2) or whether the funds are already in the UniversalRouter

PERMIT2_PERMIT

- `IAllowanceTransfer.PermitsSingle` A `PermitSingle` struct outlining the Permit2 permit to execute
- `bytes` The signature to provide to Permit2

The individual that signed the permit must be the `msg.sender` of the transaction

WRAP_ETH

- `address` The recipient of the WETH

- `uint256` The amount of ETH to wrap

UNWRAP_ETH

- `address` The recipient of the ETH
- `uint256` The minimum required ETH to receive from the unwrapping

PERMIT2_TRANSFER_FROM_BATCH

- `IAllowanceTransfer.AllowanceTransferDetails[]` An array of `AllowanceTransferDetails` structs that each describe a Permit2 transfer to perform

SEAPORT

- `uint256` The ETH value to forward to the Seaport contract
- `bytes` The calldata to use to call the Seaport contract

LOOKS_RARE_721

- `uint256` The ETH value to forward to the LooksRare contract
- `bytes` The calldata to use to call the LooksRare contract
- `address` The recipient of the ERC721
- `address` The ERC721 token address
- `uint256` The ID of the ERC721

NFTX

- `uint256` The ETH value to forward to the NFTX contract
- `bytes` The calldata to use to call the NFTX contract

CRYPTOPUNKS

- `uint256` The PunkID to purchase
- `address` The recipient for the cryptopunk
- `uint256` The ETH value to forward to the Cryptopunks contract

LOOKS_RARE_1155

- `uint256` The ETH value to forward to the LooksRare contract
- `bytes` The calldata to use to call the LooksRare contract
- `address` The recipient of the ERC1155
- `address` The ERC1155 token address
- `uint256` The ID of the ERC1155
- `uint256` The amount of the ERC1155 to transfer

OWNER_CHECK_721

- `address` The required owner of the ERC721
- `address` The ERC721 token address
- `uint256` The ID of the ERC721

OWNER_CHECK_1155

- `address` The required owner of the ERC1155
- `address` The ERC1155 token address
- `uint256` The ID of the ERC1155
- `uint256` The minimum required amount of the ERC1155

SWEEP_ERC721

- `address` The ERC721 token address to transfer
- `address` The recipient of the transfer
- `uint256` The token ID to transfer

X2Y2_721

- `uint256` The ETH value to forward to the X2Y2 contract
- `bytes` The calldata to use to call the X2Y2 contract
- `address` The recipient of the ERC721
- `address` The ERC721 token address
- `uint256` The ID of the ERC721

SUDOSWAP

- `uint256` The ETH value to forward to the Sudoswap contract
- `bytes` The calldata to use to call the Sudoswap contract

NFT20

- `uint256` The ETH value to forward to the NFT20 contract
- `bytes` The calldata to use to call the NFT20 contract

X2Y2_1155

- `uint256` The ETH value to forward to the X2Y2 contract
- `bytes` The calldata to use to call the X2Y2 contract
- `address` The recipient of the ERC1155
- `address` The ERC1155 token address
- `uint256` The ID of the ERC1155
- `uint256` The amount of the ERC1155 to transfer

FOUNDATION

- `uint256` The ETH value to forward to the Foundation contract
- `bytes` The calldata to use to call the Foundation contract
- `address` The recipient of the ERC721
- `address` The ERC721 token address
- `uint256` The ID of the ERC721

SWEEP_ERC1155

- `address` The ERC1155 token address to sweep
- `address` The recipient of the sweep
- `uint256` The token ID to sweep
- `uint256` The minimum required tokens to receive from the sweep

Example: Reverting Commands

For a Sudoswap command, that should be *allowed to revert*, the following 8 bit command should be provided:

```
command = 0x80 (10000000) && 0x19 (00011001) = 0x99 (10011001)
```

Take care when working with reverting commands - ensure you have appended commands to deal with funds that could remain in the contract after either outcomes. For example, if the Sudoswap command reverts, a following SWEEP can be added to ensure that any ETH that was not spent does not get left in the router.--- id: connect-to-uniswap title: Connect to Uniswap

The Uniswap smart contracts exist on the Ethereum blockchain. Use [ethers.js](#) or [web3.js](#) to connect your website to Ethereum. Users will need a web3-enabled browser. On desktop this means using the [MetaMask](#) extension or something similar. On mobile, web3-compatible browsers include [Trust Wallet](#) and [Coinbase Wallet](#). See [ethereum.org](#) to learn more.

Factory Contract

The Uniswap [factory contract](#) can be used to create exchange contracts for any ERC20 token that does not already have one. It also functions as a registry of ERC20 tokens that have been added to the system, and the exchange with which they are associated.

The factory contract can be instantiated using the factory address and ABI:

[Factory Address](#)

```
// mainnet
const factory = '0xc0a47dFe034B40fB47bDaD5PecDa2621de6c4d95'

// testnets
const ropsten = '0x9c83dCE8CA20E9aAF9D3efc003b2ea62aBC08351'
const rinkeby = '0xf5D915570BC477f9B8D6C0E980aA81757A3AaC36'
const kovan = '0xD3E51Ef092B2845f10401a0159B2B96e8B6c3D30'
const görli = '0x6Ce570d02D73d4c384b46135E87f8C592A8c86dA'
```

Factory Interface

Creating the factory interface in web3 requires the **factory address** and the **factory ABI**:

```
const factoryABI = [
{
  name: 'NewExchange',
  inputs: [
    { type: 'address', name: 'token', indexed: true },
    { type: 'address', name: 'exchange', indexed: true },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: 'initializeFactory',
  outputs: [],
  inputs: [{ type: 'address', name: 'template' }],
  constant: false,
  payable: false,
  type: 'function',
  gas: 35725,
},
{
  name: 'createExchange',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [{ type: 'address', name: 'token' }],
  constant: false,
  payable: false,
  type: 'function',
  gas: 187911,
},
{
  name: 'getExchange',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [{ type: 'address', name: 'token' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 715,
},
{
  name: 'getToken',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [{ type: 'address', name: 'exchange' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 745,
},
{
  name: 'getTokenWithId',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [{ type: 'uint256', name: 'token_id' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 736,
},
{
  name: 'exchangeTemplate',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 633,
},
{
  name: 'tokenCount',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
```

```

        type: 'function',
        gas: 663,
    },
]

const factoryContract = new web3.eth.Contract(factoryABI, factoryAddress)

```

Exchange Contracts

Get Exchange Address

There is a separate exchange contract for every ERC20 token. The `getExchange` method in the factory contract can be used to find the Ethereum address associated with an ERC20 token address.

```
const exchangeAddress = factoryContract.methods.getExchange(tokenAddress)
```

If the return value is `0x00` the token does not yet have an exchange.

Exchange Interface

Creating an exchange interface in web3 requires the **exchange address** and the **exchange ABI**:

```

const exchangeABI = [
{
    name: 'TokenPurchase',
    inputs: [
        { type: 'address', name: 'buyer', indexed: true },
        { type: 'uint256', name: 'eth_sold', indexed: true },
        { type: 'uint256', name: 'tokens_bought', indexed: true },
    ],
    anonymous: false,
    type: 'event',
},
{
    name: 'EthPurchase',
    inputs: [
        { type: 'address', name: 'buyer', indexed: true },
        { type: 'uint256', name: 'tokens_sold', indexed: true },
        { type: 'uint256', name: 'eth_bought', indexed: true },
    ],
    anonymous: false,
    type: 'event',
},
{
    name: 'AddLiquidity',
    inputs: [
        { type: 'address', name: 'provider', indexed: true },
        { type: 'uint256', name: 'eth_amount', indexed: true },
        { type: 'uint256', name: 'token_amount', indexed: true },
    ],
    anonymous: false,
    type: 'event',
},
{
    name: 'RemoveLiquidity',
    inputs: [
        { type: 'address', name: 'provider', indexed: true },
        { type: 'uint256', name: 'eth_amount', indexed: true },
        { type: 'uint256', name: 'token_amount', indexed: true },
    ],
    anonymous: false,
    type: 'event',
},
{
    name: 'Transfer',
    inputs: [
        { type: 'address', name: '_from', indexed: true },

```

```
{
  { type: 'address', name: '_to', indexed: true },
  { type: 'uint256', name: '_value', indexed: false },
),
anonymous: false,
type: 'event',
},
{
name: 'Approval',
inputs: [
  { type: 'address', name: '_owner', indexed: true },
  { type: 'address', name: '_spender', indexed: true },
  { type: 'uint256', name: '_value', indexed: false },
],
anonymous: false,
type: 'event',
},
{
name: 'setup',
outputs: [],
inputs: [{ type: 'address', name: 'token_addr' }],
constant: false,
payable: false,
type: 'function',
gas: 175875,
},
{
name: 'addLiquidity',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
  { type: 'uint256', name: 'min_liquidity' },
  { type: 'uint256', name: 'max_tokens' },
  { type: 'uint256', name: 'deadline' },
],
constant: false,
payable: true,
type: 'function',
gas: 82605,
},
{
name: 'removeLiquidity',
outputs: [
  { type: 'uint256', name: 'out' },
  { type: 'uint256', name: 'out' },
],
inputs: [
  { type: 'uint256', name: 'amount' },
  { type: 'uint256', name: 'min_eth' },
  { type: 'uint256', name: 'min_tokens' },
  { type: 'uint256', name: 'deadline' },
],
constant: false,
payable: false,
type: 'function',
gas: 116814,
},
{
name: '__default__',
outputs: [],
inputs: [],
constant: false,
payable: true,
type: 'function',
},
{
name: 'ethToTokenSwapInput',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
  { type: 'uint256', name: 'min_tokens' },
  { type: 'uint256', name: 'deadline' },
],
},
```

```
constant: false,
payable: true,
type: 'function',
gas: 12757,
},
{
name: 'ethToTokenTransferInput',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
{ type: 'uint256', name: 'min_tokens' },
{ type: 'uint256', name: 'deadline' },
{ type: 'address', name: 'recipient' },
],
constant: false,
payable: true,
type: 'function',
gas: 12965,
},
{
name: 'ethToTokenSwapOutput',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
{ type: 'uint256', name: 'tokens_bought' },
{ type: 'uint256', name: 'deadline' },
],
constant: false,
payable: true,
type: 'function',
gas: 50455,
},
{
name: 'ethToTokenTransferOutput',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
{ type: 'uint256', name: 'tokens_bought' },
{ type: 'uint256', name: 'deadline' },
{ type: 'address', name: 'recipient' },
],
constant: false,
payable: true,
type: 'function',
gas: 50663,
},
{
name: 'tokenToEthSwapInput',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
{ type: 'uint256', name: 'tokens_sold' },
{ type: 'uint256', name: 'min_eth' },
{ type: 'uint256', name: 'deadline' },
],
constant: false,
payable: false,
type: 'function',
gas: 47503,
},
{
name: 'tokenToEthTransferInput',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
{ type: 'uint256', name: 'tokens_sold' },
{ type: 'uint256', name: 'min_eth' },
{ type: 'uint256', name: 'deadline' },
{ type: 'address', name: 'recipient' },
],
constant: false,
payable: false,
type: 'function',
gas: 47712,
},
```

```
{
  name: 'tokenToEthSwapOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'eth_bought' },
    { type: 'uint256', name: 'max_tokens' },
    { type: 'uint256', name: 'deadline' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 50175,
},
{
  name: 'tokenToEthTransferOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'eth_bought' },
    { type: 'uint256', name: 'max_tokens' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 50384,
},
{
  name: 'tokenToTokenSwapInput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_sold' },
    { type: 'uint256', name: 'min_tokens_bought' },
    { type: 'uint256', name: 'min_eth_bought' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'token_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 51007,
},
{
  name: 'tokenToTokenTransferInput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_sold' },
    { type: 'uint256', name: 'min_tokens_bought' },
    { type: 'uint256', name: 'min_eth_bought' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
    { type: 'address', name: 'token_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 51098,
},
{
  name: 'tokenToTokenSwapOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_bought' },
    { type: 'uint256', name: 'max_tokens_sold' },
    { type: 'uint256', name: 'max_eth_sold' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'token_addr' },
  ],
  constant: false,
  payable: false,
}
```

```
        type: 'function',
        gas: 54928,
    },
    {
        name: 'tokenToTokenTransferOutput',
        outputs: [{ type: 'uint256', name: 'out' }],
        inputs: [
            { type: 'uint256', name: 'tokens_bought' },
            { type: 'uint256', name: 'max_tokens_sold' },
            { type: 'uint256', name: 'max_eth_sold' },
            { type: 'uint256', name: 'deadline' },
            { type: 'address', name: 'recipient' },
            { type: 'address', name: 'token_addr' },
        ],
        constant: false,
        payable: false,
        type: 'function',
        gas: 55019,
    },
    {
        name: 'tokenToExchangeSwapInput',
        outputs: [{ type: 'uint256', name: 'out' }],
        inputs: [
            { type: 'uint256', name: 'tokens_sold' },
            { type: 'uint256', name: 'min_tokens_bought' },
            { type: 'uint256', name: 'min_eth_bought' },
            { type: 'uint256', name: 'deadline' },
            { type: 'address', name: 'exchange_addr' },
        ],
        constant: false,
        payable: false,
        type: 'function',
        gas: 49342,
    },
    {
        name: 'tokenToExchangeTransferInput',
        outputs: [{ type: 'uint256', name: 'out' }],
        inputs: [
            { type: 'uint256', name: 'tokens_sold' },
            { type: 'uint256', name: 'min_tokens_bought' },
            { type: 'uint256', name: 'min_eth_bought' },
            { type: 'uint256', name: 'deadline' },
            { type: 'address', name: 'recipient' },
            { type: 'address', name: 'exchange_addr' },
        ],
        constant: false,
        payable: false,
        type: 'function',
        gas: 49532,
    },
    {
        name: 'tokenToExchangeSwapOutput',
        outputs: [{ type: 'uint256', name: 'out' }],
        inputs: [
            { type: 'uint256', name: 'tokens_bought' },
            { type: 'uint256', name: 'max_tokens_sold' },
            { type: 'uint256', name: 'max_eth_sold' },
            { type: 'uint256', name: 'deadline' },
            { type: 'address', name: 'exchange_addr' },
        ],
        constant: false,
        payable: false,
        type: 'function',
        gas: 53233,
    },
    {
        name: 'tokenToExchangeTransferOutput',
        outputs: [{ type: 'uint256', name: 'out' }],
        inputs: [
            { type: 'uint256', name: 'tokens_bought' },
```

```
{ type: 'uint256', name: 'max_tokens_sold' },
{ type: 'uint256', name: 'max_eth_sold' },
{ type: 'uint256', name: 'deadline' },
{ type: 'address', name: 'recipient' },
{ type: 'address', name: 'exchange_addr' },
],
constant: false,
payable: false,
type: 'function',
gas: 53423,
},
{
name: 'getEthToTokenInputPrice',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [{ type: 'uint256', name: 'eth_sold' }],
constant: true,
payable: false,
type: 'function',
gas: 5542,
},
{
name: 'getEthToTokenOutputPrice',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [{ type: 'uint256', name: 'tokens_bought' }],
constant: true,
payable: false,
type: 'function',
gas: 6872,
},
{
name: 'getTokenToEthInputPrice',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [{ type: 'uint256', name: 'tokens_sold' }],
constant: true,
payable: false,
type: 'function',
gas: 5637,
},
{
name: 'getTokenToEthOutputPrice',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [{ type: 'uint256', name: 'eth_bought' }],
constant: true,
payable: false,
type: 'function',
gas: 6897,
},
{
name: 'tokenAddress',
outputs: [{ type: 'address', name: 'out' }],
inputs: [],
constant: true,
payable: false,
type: 'function',
gas: 1413,
},
{
name: 'factoryAddress',
outputs: [{ type: 'address', name: 'out' }],
inputs: [],
constant: true,
payable: false,
type: 'function',
gas: 1443,
},
{
name: 'balanceOf',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [{ type: 'address', name: '_owner' }],
constant: true,
```

```
payable: false,
type: 'function',
gas: 1645,
},
{
name: 'transfer',
outputs: [{ type: 'bool', name: 'out' }],
inputs: [
{ type: 'address', name: '_to' },
{ type: 'uint256', name: '_value' },
],
constant: false,
payable: false,
type: 'function',
gas: 75034,
},
{
name: 'transferFrom',
outputs: [{ type: 'bool', name: 'out' }],
inputs: [
{ type: 'address', name: '_from' },
{ type: 'address', name: '_to' },
{ type: 'uint256', name: '_value' },
],
constant: false,
payable: false,
type: 'function',
gas: 110907,
},
{
name: 'approve',
outputs: [{ type: 'bool', name: 'out' }],
inputs: [
{ type: 'address', name: '_spender' },
{ type: 'uint256', name: '_value' },
],
constant: false,
payable: false,
type: 'function',
gas: 38769,
},
{
name: 'allowance',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
{ type: 'address', name: '_owner' },
{ type: 'address', name: '_spender' },
],
constant: true,
payable: false,
type: 'function',
gas: 1925,
},
{
name: 'name',
outputs: [{ type: 'bytes32', name: 'out' }],
inputs: [],
constant: true,
payable: false,
type: 'function',
gas: 1623,
},
{
name: 'symbol',
outputs: [{ type: 'bytes32', name: 'out' }],
inputs: [],
constant: true,
payable: false,
type: 'function',
gas: 1653,
```

```

},
{
  name: 'decimals',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 1683,
},
{
  name: 'totalSupply',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 1713,
},
]

```

```
const exchangeContract = new web3.eth.Contract(exchangeABI, exchangeAddress)
```

Token Contracts

Some Uniswap interactions require making calls directly to ERC20 token contracts rather than the exchanges with which they are associated.

Get Token Address

The `getToken` method in the factory contract can be used to find the ERC20 token address associated with an exchange contract. There is no barrier of entry for adding an ERC20 token to Uniswap or checks on the validity of the token contracts. Frontend interfaces should maintain a list of valid ERC20 tokens that users can safely trade or allow users to paste in arbitrary addresses.

```
const tokenAddress = factoryContract.methods.getToken(exchangeAddress)
```

If the return value is `0x00` the input address is not a Uniswap exchange.

Token Interface

Creating a token interface in web3 requires the **token address** and the **token ABI**:

```

const tokenABI = [
{
  name: 'Transfer',
  inputs: [
    { type: 'address', name: '_from', indexed: true },
    { type: 'address', name: '_to', indexed: true },
    { type: 'uint256', name: '_value', indexed: false },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: 'Approval',
  inputs: [
    { type: 'address', name: '_owner', indexed: true },
    { type: 'address', name: '_spender', indexed: true },
    { type: 'uint256', name: '_value', indexed: false },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: '__init__',
  outputs: [],
  inputs: [
    { type: 'bytes32', name: '_name' },
  ]
}
]
```

```
{
  { type: 'bytes32', name: '_symbol' },
  { type: 'uint256', name: '_decimals' },
  { type: 'uint256', name: '_supply' },
},
constant: false,
payable: false,
type: 'constructor',
},
{
name: 'deposit',
outputs: [],
inputs: [],
constant: false,
payable: true,
type: 'function',
gas: 74279,
},
{
name: 'withdraw',
outputs: [{ type: 'bool', name: 'out' }],
inputs: [{ type: 'uint256', name: '_value' }],
constant: false,
payable: false,
type: 'function',
gas: 108706,
},
{
name: 'totalSupply',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [],
constant: true,
payable: false,
type: 'function',
gas: 543,
},
{
name: 'balanceOf',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [{ type: 'address', name: '_owner' }],
constant: true,
payable: false,
type: 'function',
gas: 745,
},
{
name: 'transfer',
outputs: [{ type: 'bool', name: 'out' }],
inputs: [
  { type: 'address', name: '_to' },
  { type: 'uint256', name: '_value' },
],
constant: false,
payable: false,
type: 'function',
gas: 74698,
},
{
name: 'transferFrom',
outputs: [{ type: 'bool', name: 'out' }],
inputs: [
  { type: 'address', name: '_from' },
  { type: 'address', name: '_to' },
  { type: 'uint256', name: '_value' },
],
constant: false,
payable: false,
type: 'function',
gas: 110600,
},
```

```

name: 'approve',
outputs: [{ type: 'bool', name: 'out' }],
inputs: [
  { type: 'address', name: '_spender' },
  { type: 'uint256', name: '_value' },
],
constant: false,
payable: false,
type: 'function',
gas: 37888,
},
{
  name: 'allowance',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [
  { type: 'address', name: '_owner' },
  { type: 'address', name: '_spender' },
],
constant: true,
payable: false,
type: 'function',
gas: 1025,
},
{
  name: 'name',
outputs: [{ type: 'bytes32', name: 'out' }],
inputs: [],
constant: true,
payable: false,
type: 'function',
gas: 723,
},
{
  name: 'symbol',
outputs: [{ type: 'bytes32', name: 'out' }],
inputs: [],
constant: true,
payable: false,
type: 'function',
gas: 753,
},
{
  name: 'decimals',
outputs: [{ type: 'uint256', name: 'out' }],
inputs: [],
constant: true,
payable: false,
type: 'function',
gas: 783,
},
]

```

```
const tokenContract = new web3.eth.Contract(tokenABI, tokenAddress)
```

id: pool-liquidity title: Pool Liquidity

Formalized Model

Uniswap liquidity pools are autonomous and use the Constant Product Market Maker ($x * y = k$). This model was formalized and the smart contract implementation passed a lightweight formal verification.

- [Formalized Specification](#)
- [Lightweight Verification](#)

Create Exchange

The `createExchange` function is used to deploy exchange contracts for ERC20 tokens that do not yet have one.

```
factory.methods.createExchange(tokenAddress).send()
```

Once an exchange is created the address can be retrieved with `getExchange`.

Exchange Reserves

Each exchange contract holds a liquidity reserve of ETH and its associated ERC20 token.

ETH Reserve

The ETH reserve associated with an ERC20 token exchange is the ETH balance of the exchange smart contract.

```
const ethReserve = web3.eth.getBalance(exchangeAddress)
```

ERC20 Reserve

The ERC20 reserve associated with an ERC20 token exchange is the ERC20 balance of the exchange smart contract.

```
const tokenReserve = tokenContract.methods.balanceOf(exchangeAddress)
```

Add Liquidity

Anyone who wants can join a Uniswap liquidity pool by calling the `addLiquidity` function.

```
exchange.methods.addLiquidity(min_liquidity, max_tokens, deadline).send({ value: ethAmount })
```

Adding liquidity requires depositing an equivalent `value` of ETH and ERC20 tokens into the ERC20 token's associated exchange contract.

The first liquidity provider to join a pool sets the initial exchange rate by depositing what they believe to be an equivalent value of ETH and ERC20 tokens. If this ratio is off, arbitrage traders will bring the prices to equilibrium at the expense of the initial liquidity provider.

All future liquidity providers deposit ETH and ERC20's using the exchange rate at the moment of their deposit. If the exchange rate is bad there is a profitable arbitrage opportunity that will correct the price.

Parameters

The `ethAmount` sent to `addLiquidity` is the exact amount of ETH that will be deposited into the liquidity reserves. It should be 50% of the total value a liquidity provider wishes to deposit into the reserves.

Since liquidity providers must deposit at the current exchange rate, the Uniswap smart contracts use `ethAmount` to determine the amount of ERC20 tokens that must be deposited. This token amount is the remaining 50% of total value a liquidity provider wishes to deposit. Since exchange rate can change between when a transaction is signed and when it is executed on Ethereum, `max_tokens` is used to bound the amount this rate can fluctuate. For the first liquidity provider, `max_tokens` is the exact amount of tokens deposited.

Liquidity tokens are minted to track the relative proportion of total reserves that each liquidity provider has contributed. `min_liquidity` is used in combination with `max_tokens` and `ethAmount` to bound the rate at which liquidity tokens are minted. For the first liquidity provider, `min_liquidity` does not do anything and can be set to 0.

Transaction `deadline` is used to set a time after which a transaction can no longer be executed. This limits the "free option" problem, where Ethereum miners can hold signed transactions and execute them based off market movements.

Remove Liquidity

Liquidity providers use the `removeLiquidity` function to withdraw their portion of the reserves.

```
exchange.methods.removeLiquidity(amount, min_eth, min_tokens, deadline).send()
```

Liquidity is withdrawn at the same ratio as the reserves at the time of withdrawal. If the exchange rate is bad there is a profitable arbitrage opportunity that will correct the price.

Parameters

`amount` specifies the number of liquidity tokens that will be burned. Dividing this amount by the total liquidity token supply gives the percentage of both the ETH and ER20 reserves the provider is withdrawing.

Since exchange rate can change between when a transaction is signed and when it is executed on Ethereum, `min_eth` and `min_tokens` are used to bound the amount this rate can fluctuate.

Same as in `addLiquidity`, `deadline` is used to set a time after which a transaction can no longer be executed.

id: trade-tokens title: Trade Tokens

In Uniswap, there is a separate exchange contract for each ERC20 token. These exchanges hold reserves of both ETH and their associated ERC20. Instead of waiting to be matched in an order-book, users can make trades against the reserves at any time. Reserves are pooled between a decentralized network of liquidity providers who collect fees on every trade.

Pricing is automatic, based on the $x * y = k$ market making formula which automatically adjusts prices based off the relative sizes of the two reserves and the size of the incoming trade. Since all tokens share ETH as a common pair, it is used as an intermediary asset for direct trading between any ERC20 \rightleftharpoons ERC20 pair.

ETH \rightleftharpoons ERC20 Calculations

The variables needed to determine price when trading between ETH and ERC20 tokens is:

- ETH reserve size of the ERC20 exchange
- ERC20 reserve size of the ERC20 exchange
- Amount sold (input) or amount bought (output)

Amount Bought (sell order)

For sell orders (exact input), the amount bought (output) is calculated:

```
// Sell ETH for ERC20
const inputAmount = userInputEthValue
const inputReserve = web3.eth.getBalance(exchangeAddress)
const outputReserve = tokenContract.methods.balanceOf(exchangeAddress).call()

// Sell ERC20 for ETH
const inputAmount = userInputTokenValue
const inputReserve = tokenContract.methods.balanceOf(exchangeAddress).call()
const outputReserve = web3.eth.getBalance(exchangeAddress)

// Output amount bought
const numerator = inputAmount * outputReserve * 997
const denominator = inputReserve * 1000 + inputAmount * 997
const outputAmount = numerator / denominator
```

Amount Sold (buy order)

For buy orders (exact output), the cost (input) is calculated:

```
// Buy ERC20 with ETH
const outputAmount = userInputTokenValue
const inputReserve = web3.eth.getBalance(exchangeAddress)
const outputReserve = tokenContract.methods.balanceOf(exchangeAddress).call()

// Buy ETH with ERC20
const outputAmount = userInputEthValue
const inputReserve = tokenContract.methods.balanceOf(exchangeAddress).call()
const outputReserve = web3.eth.getBalance(exchangeAddress)

// Cost
const numerator = outputAmount * inputReserve * 1000
const denominator = (outputReserve - outputAmount) * 997
const inputAmount = numerator / denominator + 1
```

Liquidity Provider Fee

There is a 0.3% liquidity provider fee built into the price formula. This can be calculated:

```
fee = inputAmount * 0.003
```

Exchange Rate

The exchange rate is simply the output amount divided by the input amount.

```
const rate = outputAmount / inputAmount
```

ERC20 ⇌ ERC20 Calculations

The variables needed to determine price when trading between two ERC20 tokens is:

- ETH reserve size of the input ERC20 exchange
- ERC20 reserve size of the input ERC20 exchange
- ETH reserve size of the output ERC20 exchange
- ERC20 reserve size of the output ERC20 exchange
- Amount sold (input) or amount bought (output)

Amount Bought (sell order)

For sell orders (exact input), the amount bought (output) is calculated:

```
// TokenA (ERC20) to ETH conversion
const inputAmountA = userInputTokenAValue
const inputReserveA = tokenContractA.methods.balanceOf(exchangeAddressA).call()
const outputReserveA = web3.eth.getBalance(exchangeAddressA)

const numeratorA = inputAmountA * outputReserveA * 997
const denominatorA = inputReserveA * 1000 + inputAmountA * 997
const outputAmountA = numeratorA / denominatorA

// ETH to TokenB conversion
const inputAmountB = outputAmountA
const inputReserveB = web3.eth.getBalance(exchangeAddressB)
const outputReserveB = tokenContractB.methods.balanceOf(exchangeAddressB).call()

const numeratorB = inputAmountB * outputReserveB * 997
const denominatorB = inputReserveB * 1000 + inputAmountB * 997
const outputAmountB = numeratorB / denominatorB
```

Amount Sold (buy order)

For buy orders (exact output), the cost (input) is calculated:

```
// Buy TokenB with ETH
const outputAmountB = userInputTokenBValue
const inputReserveB = web3.eth.getBalance(exchangeAddressB)
const outputReserveB = tokenContractB.methods.balanceOf(exchangeAddressB).call()

// Cost
const numeratorB = outputAmountB * inputReserveB * 1000
const denominatorB = (outputReserveB - outputAmountB) * 997
const inputAmountB = numeratorB / denominatorB + 1

// Buy ETH with TokenA
const outputAmountA = userInputEthValue
const inputReserveA = tokenContractA.methods.balanceOf(exchangeAddressA).call()
const outputReserveA = web3.eth.getBalance(exchangeAddressA)

// Cost
const numeratorA = outputAmountA * inputReserveA * 1000
const denominatorA = (outputReserveA - outputAmountA) * 997
const inputAmountA = numeratorA / denominatorA + 1
```

Liquidity Provider Fee

There is a 0.30% liquidity provider fee to swap from TokenA to ETH on the input exchange. There is another 0.3% liquidity provider fee to swap the remaining ETH to TokenB.

```
const exchangeAFee = inputAmountA * 0.003
const exchangeBFee = inputAmountB * 0.003
```

Since users only inputs Token A, it can be represented to them as:

```
const combinedFee = inputAmountA * 0.00591
```

Exchange Rate

The exchange rate is simply the output amount divided by the input amount.

```
const rate = outputAmountB / inputAmountA
```

Deadlines

Many Uniswap functions include a transaction `deadline` that sets a time after which a transaction can no longer be executed. This limits miners holding signed transactions for extended durations and executing them based off market movements. It also reduces uncertainty around transactions that take a long time to execute due to issues with gas price.

Deadlines are calculated by adding the desired amount of time (in seconds) to the latest Ethereum block timestamp.

```
web3.eth.getBlock('latest', (error, block) => {
  deadline = block.timestamp + 300 // transaction expires in 300 seconds (5 minutes)
})
```

Recipients

Uniswap allows traders to swap tokens and transfer the output to a new `recipient` address. This allows for a type of payment where the payer sends one token and the payee receives another.

ETH ⇌ ERC20 Trades

Coming soon...

ERC20 ⇌ ERC20 Trades

Coming soon...

Custom Pools

Coming soon...

id: custom-linking title: Custom Linking

Query Parameters

The Uniswap front-end supports URL query parameters to allow for custom linking to the Uniswap exchange. Users and developers can use these query parameters to link to the Uniswap exchange with custom prefilled settings.

Each Page has specific available URL parameters that can be set. Global parameters can be used on all pages.

A parameter used on an incorrect page will have no effect on exchange settings. Parameters not set with a URL parameter will be set to standard exchange defaults.

Global

Parameter	Type	Description
theme	String	Sets them to dark or light mode.

Theme Options

Theme can be set as `light` or `dark`.

Example Usage

```
https://app.uniswap.org/#/swap?theme=dark&use=v1
```

Swap Page

Parameter	Type	Description
-----------	------	-------------

inputCurrency	address	Input currency that will be swapped for output currency.
outputCurrency	address or ETH	Output currency that input currency will be swapped for.
slippage	number	Max slippage to be used during transaction (in bips)
exactAmount	number	The custom token amount to buy or sell.
exactField	string	The field to set custom token amount for. Must be <code>input</code> or <code>output</code> .

Defaults

ETH defaults as the input currency. When a different token is selected for either input or output ETH will default as the opposite selected currency.

Constraints

Addresses must be valid ERC20 addresses. Slippage and amount values must be valid numbers accepted by the exchange (or error will prevent from swapping). Slippage can 0, or within the range 10->9999 bips (which converts to 0%, 0.01%->99%)

When selecting ETH as the output currency a user must also choose an inputCurrency that is not ETH (to prevent ETH being populated in both fields)

Setting Amounts

Two parameters, exactField and exactAmount can be used to set specific token amounts to be sold or bought. Both fields must be set in the URL or there will be no effect on the settings.

Example Usage

```
https://app.uniswap.org/#/swap?  
exactField=input&exactAmount=10&inputCurrency=0x0F5D2fB29fb7d3CFeE444a200298f468908cC942?use=v1
```

Send Page

The send page has the same options available as the Swap page, plus one additional parameter, `recipient`.

Parameter	Type	Description
recipient	address	Address of the recipient of a send transaction.

Example Usage

```
https://app.uniswap.org/#/send?recipient=0x74Aa01d162E6dC6A657caC857418C403D48E2D77?use=v1
```

Pool Page

The Pool page is made up of 3 subroutines: `add-liquidity`, `remove-liquidity`, `create-exchange`.

Add Liquidity

Parameter	Type	Description
ethAmount	number	Amount of ETH to deposit into the pool.
token	address	ERC20 address of the pool to add liquidity to.
tokenAmount	number	Amount of the selected token to deposit into the pool.

Example Usage

```
https://app.uniswap.org/#/add-liquidity?  
ethAmount=2.34&token=0x42456D7084eacF4083f1140d3229471bbA2949A8&tokenAmount=300?use=v1
```

Remove Liquidity

Parameter	Type	Description
poolTokenAddress	address	Pool to withdraw liquidity from. (Must be an ERC20 address with an existing exchange)
poolTokenAmount	number	Amount of pool token to be withdrawn from liquidity pool.

Example Usage

<https://app.uniswap.org/#/remove-liquidity?poolTokenAmount=1.23&use=v1>

Create Exchange

Parameter	Type	Description
tokenAddress	address	ERC20 token to create the exchange for. Must be valid ERC20 token for which there is no existing exchange.

Example Usage

<https://app.uniswap.org/#/swap?use=v1&create-exchange?tokenAddress=0x0F5D2fB29fb7d3CFeE444a200298f468908cc942>

Custom Routes

Custom token routes can still be used in combination with URL parameters. URL parameters are higher in the settings hierarchy than custom routes.

An example using custom token route and URL parameters.

**https://app.uniswap.org/#/swap/0x0F5D2fB29fb7d3CFeE444a200298f468908cc942?
exactField=input&exactAmount=10&use=v1**

id: iframe-integration title: Iframe Integration

Uniswap can be used within other sites as an iframe. An iframe shows an exact version of the app.uniswap.org site and can have custom prefilled settings.

Why You May Want This

Integrating the Uniswap site directly into your web application can be useful for a variety of reasons.

v1.app.uniswap.org allows users to buy, sell, send, or provide liquidity for ERC20 tokens. An iframe integration may be useful if your application provides services around these ERC20 tokens. (For example, users can buy DAI through a Uniswap iframe on your site, then allow users to lend that DAI on your site).

It can also be useful if your application requires users to acquire some token in order to use some service (For example, allow users to buy "REP" token so they can engage in prediction markets on the Augur Dapp).

iframe vs. custom UI

One benefit of an iframe integration is that your site will automatically keep up with any improvements/additions to the v1.app.uniswap.org site. After the initial integration is setup no further work is needed to pull in updates as the exchange site is updated over time.

Live Example

An example of an Iframe integration can be found on the FOAM site <https://map.foam.space/>

To see the Iframe click the dropdown in the top right and click "get foam".

Add To Your Site

To include a Uniswap iframe within your site just add an iframe element within your website code and link to the Uniswap exchange.

Linking to a ETH <-> DAI swap page would look something like this. To link to a token of your choice replace the address after "outputCurrency" with the token address of the token you want to link to.

```
<iframe  
src="https://app.uniswap.org/#/swap?use=v1?outputCurrency=0x89d24a6b4ccb1b6faa2625fe562bdd9a23260359"  
height="660px"  
width="100%"  
style=""  
border: 0;  
margin: 0 auto;  
display: block;  
border-radius: 10px;  
max-width: 600px;  
min-width: 300px;  
">
```

```
    id="myId"  
  />
```

You can customize the selected page, selected custom tokens and more using URL query parameters. See [Custom Linking](#).

id: token-listing title: Token Listing

It is possible that a token you are interested in is not included in the token dropdown on <https://app.uniswap.org/#/swap?use=v1>, however, all tokens that have a deployed uniswap exchange are supported on the front-end.

There are three ways to interact with tokens that are not yet included on the default list.

1. Paste the token address into the search box.

If a token is not included in the list, try pasting the token address into the search box. It will populate the dropdown with the token you are looking for.

2. Custom Linking

<https://app.uniswap.org/#/swap?use=v1> supports custom linking to all tokens that have a Uniswap exchange. See [Custom Linking](#) for details on how to link.

For example, to populate the output token field with an unlisted token, we can specify the outputCurrency in the URL and pass in the token's address like this:

```
https://app.uniswap.org/#/swap?use=v1?outputCurrency=0xfa3e941d1f6b7b10eD84A0C211bfA8aeE90796e
```

Token Details and Assets

Token information (including decimals, symbol, name, etc.) is pulled from token contracts directly. Logo images are pulled from TrustWallet. If you'd like your token logo updated make a pull request into the TrustWallet assets repo <https://github.com/trustwallet/assets>.

id: overview title: Overview sidebar_position: 1

The Uniswap V1 Smart Contracts

Uniswap V1 is the first version of the protocol, [launched in November 2018](#) at Devcon 4. Because of its permissionless nature, it will exist for as long as Ethereum does.

Designed with simplicity in mind, the Uniswap protocol provides an interface for seamless exchange of ERC20 tokens on Ethereum. By eliminating unnecessary forms of rent extraction and middlemen it allows faster, more efficient exchange. Where it makes tradeoffs, decentralization, censorship resistance, and security are prioritized.

Uniswap is open source and functions as a public good. There is no central token or platform fee. No special treatment is given to early investors, adopters, or developers. Token listing is open and free. All smart contract functions are public and all upgrades are opt-in.

This site will serve as a project overview for Uniswap - explaining how it works, how to use it, and how to build on top of it. These docs are actively being worked on and more information will be added on an ongoing basis.

V1 Features

- Add support for any ERC20 token using the Uniswap [factory](#).
- Join liquidity pools to collect fees on ETH-ERC20 pairs
- Liquidity-sensitive automated pricing using [constant product formula](#)
- Trade ETH for any ERC20 without wrapping
- Trade any ERC20 for any ERC20 in a single transaction
- Trade and transfer to a different address in a single transaction
- Lowest gas cost of any decentralized exchange
- Support for private and custom uniswap exchanges
- Buy ERC20 tokens from any wallet using ENS
- [Partially verified](#) smart contracts written in Vyper
- Mobile-optimized open source [frontend implementation](#)
- Funded through an [Ethereum Foundation grant](#)

Resources

- [Website](#)
- [GitHub](#)
- [Twitter](#)
- [Reddit](#)
- [Email](#)
- [Whitepaper](#)

How it works

Uniswap is made up of a series of ETH-ERC20 exchange contracts. There is exactly one exchange contract per ERC20 token. If a token does not yet have an exchange it can be created by anyone using the Uniswap factory contract. The factory serves as a public registry and is used to look up all token and exchange addresses added to the system.

Each exchange holds reserves of both ETH and its associated ERC20 token. Anyone can become a liquidity provider on an exchange and contribute to its reserves. This is different than buying or selling; it requires depositing an equivalent value of both ETH and the relevant ERC20 token. Liquidity is pooled across all providers and an internal "pool token" (ERC20) is used to track each providers relative contribution. Pool tokens are minted when liquidity is deposited into the system and can be burned at any time to withdraw a proportional share of the reserves.

Exchange contracts are automated market makers between an ETH-ERC20 pair. Traders can swap between the two in either direction by adding to the liquidity reserve of one and withdrawing from the reserve of the other. Since ETH is a common pair for all ERC20 exchanges, it can be used as an intermediary allowing direct ERC20-ERC20 trades in a single transaction. Users can specify a recipient address if they want to receive purchased tokens at a different address from the one used to make a transaction.

Uniswap uses a "constant product" market making formula which sets the exchange rate based off of the relative size of the ETH and ERC20 reserves, and the amount with which an incoming trade shifts this ratio. Selling ETH for ERC20 tokens increases the size of the ETH reserve and decreases the size of the ERC20 reserve. This shifts the reserve ratio, increasing the ERC20 token's price relative to ETH for subsequent transactions. The larger a trade relative to the total size of the reserves, the more price slippage will occur. Essentially, exchange contracts use the open financial market to decide on the relative value of a pair and uses that as a market making strategy.

A small liquidity provider fee (0.30%) is taken out of each trade and added to the reserves. While the ETH-ERC20 reserve ratio is constantly shifting, fees makes sure that the total combined reserve size increases with every trade. This functions as a payout to liquidity providers that is collected when they burn their pool tokens to withdraw their portion of total reserves. Guaranteed arbitrage opportunities from price fluctuations should push a steady flow of transactions through the system and increase the amount of fee revenue generated.

Since Uniswap is entirely on-chain, prices can change between when a transaction is signed and when it is included in a block. Traders can bound price fluctuations by specifying the minimum amount bought on sell orders, or the maximum amount sold on buy orders. This acts as a limit order that will automatically cancel if it is not filled. It is also possible to set transaction deadlines which will cancel orders if they are not executed fast enough.

The reason only one exchange per token can be registered to the factory is to encourage providers to pool their liquidity into a single reserve. However, Uniswap has built in support for ERC20-to-ERC20 trades using the public pools from the factory on one side of the transaction and custom, user-specified pool on the other. Custom pools could have fund managers, use alternate pricing mechanisms, remove liquidity provider fees, integrate complex three dimensional fomo-based ponzi-schemes and more. They just need to implement the Uniswap interface and accept ETH as an intermediary asset. Custom pools do not have the same safety properties as the public ones. It is recommended users only interact with audited, open-source smart contracts.

Upgrading censorship resistant, decentralized smart contracts is difficult. If significant improvements are made to the system a new version will be released. Liquidity providers can choose between moving to the new system or staying in the old one. If possible, new versions will be backwards compatible and able to trade ERC20-to-ERC20 with the old versions similar to a custom pool.

How to use it

uniswap.org is the landing page for the Uniswap protocol. It describes the project and directs users where they need to go.

The Uniswap smart contracts live on Ethereum. Anyone can interact with them directly.

The Uniswap frontend is an open source interface designed to improve user experience when interacting with the smart contracts. Anyone can use the source code to host an interface, or build their own. Hosted interfaces are independent of Uniswap, and should comply with their jurisdictional laws and regulations.

id: factory title: Factory

initializeFactory

Parameter	Description
template	Ethereum address of exchange template

Smart Contract

```
initializeFactory(template: address)
```

Web3

```
factoryContract.methods.initializeFactory(template).send()
```

createExchange

Parameter	Type	Description
token	address	Ethereum address of an ERC20 token

Returns	
address	Ethereum address of a Uniswap exchange

Smart Contract

```
createExchange(token: address): address
```

Web3

```
factoryContract.methods.createExchange(token).send()
```

getExchange

Parameter	Type	Description
token	address	Ethereum address of an ERC20 token

Returns	
address	Ethereum address of a Uniswap exchange

Smart Contract

```
@constant
getExchange(token: address): address
```

Web3

```
factoryContract.methods.getExchange(token).call()
```

getToken

Parameter	Type	Description
exchange	address	Ethereum address of a Uniswap exchange

Returns	
---------	--

address	Ethereum address of an ERC20 token
---------	------------------------------------

Smart Contract

```
@constant  
getToken(exchange: address): address
```

Web3

```
factoryContract.methods.getToken(exchange).call()
```

getTokenWithId

Parameter	Type	Description
token_id	uint256	Uniswap ID for an ERC20 token

Returns	
address	Ethereum address of an ERC20 token

Smart Contract

```
@constant  
getTokenWithId(token_id: uint256): address
```

Web3

```
factoryContract.methods.getTokenWithId(token_id).call()
```

id: exchange title: Exchange

setup

Parameter	Description
token_addr	Ethereum address of an ERC20 Token

Smart Contract

```
# Can only be called by factory contract during createExchange()  
setup(token_addr: address):
```

Web3

```
// Can only be called by factory contract during createExchange()  
exchangeContract.methods.setup((token: String)).send()
```

addLiquidity

Parameter	Type	Description
msg.value	uint256	Amount of ETH added
min_liquidity	uint256	Minimum minted liquidity

max_tokens	uint256	Maximum ERC20 tokens added
deadline	uint256	Transaction deadline

Returns	
uint256	Amount of liquidity tokens minted

Smart Contract

```
@payable
addLiquidity(
    min_liquidity: uint256,
    max_tokens: uint256,
    deadline: uint256
): uint256
```

Web3

```
exchangeContract.methods.addLiquidity(min_liquidity, max_tokens, deadline).send({ value: ethValue })
```

removeLiquidity

Parameter	Type	Description
amount	uint256	Amount of liquidity burned
min_eth	uint256	Minimum ETH removed
min_tokens	uint256	Minimum ERC20 tokens removed
deadline	uint256	Transaction deadline

Returns	
uint256	Amount of ETH removed
uint256	Amount of ERC20 tokens removed.

Smart Contract

```
removeLiquidity(
    amount: uint256;
    min_eth: uint256,
    min_tokens: uint256,
    deadline: uint256
): (uint256, uint256)
```

Web3

```
exchangeContract.methods.removeLiquidity(amount, min_eth, min_tokens, deadline).send()
```

default

Parameter	Type	Description
msg.value	uint256	Amount of ETH sold

Smart Contract

```
# Default function in Vyper replaces the "fallback" function in Solidity
@payable
__default__():


```

Web3

```
web3.eth.sendTransaction({ value: ethAmount })
```

ethToTokenSwapInput

Parameter	Type	Description
msg.value	uint256	Amount of ETH sold
min_tokens	uint256	Minimum ERC20 tokens bought
deadline	uint256	Transaction deadline
Returns		
uint256	Amount of ERC20 tokens bought	

Smart Contract

```
@payable
ethToTokenSwapInput(
    min_tokens: uint256,
    deadline: uint256
): uint256
```

Web3

```
exchangeContract.methods.ethToTokenSwapInput(min_liquidity, max_tokens, deadline).send({ value: ethValue })
```

ethToTokenTransferInput

Parameter	Type	Description
msg.value	uint256	Amount of ETH sold
min_tokens	uint256	Minimum ERC20 tokens bought
deadline	uint256	Transaction deadline
recipient	address	Address that receives ERC20 tokens
Returns		
uint256	Amount of ERC20 tokens bought	

Smart Contract

```
@payable
ethToTokenTransferInput(
    min_tokens: uint256,
    deadline: uint256,
    recipient: address
): uint256
```

Web3

```
exchangeContract.methods
  .ethToTokenTransferInput(min_liquidity, max_tokens, deadline, recipient)
  .send({ value: ethValue })
```

ethToTokenSwapOutput

Parameter	Type	Description
msg.value	uint256	Maximum ETH sold
tokens_bought	uint256	Amount of ERC20 tokens bought
deadline	uint256	Transaction deadline

Returns	
uint256	Amount of ETH sold

Smart Contract

```
@payable
ethToTokenSwapOutput(
  tokens_bought: uint256,
  deadline: uint256
): uint256
```

Web3

```
exchangeContract.methods.ethToTokenSwapOutput(tokens_bought, deadline).send({ value: ethValue })
```

ethToTokenTransferOutput

Parameter	Type	Description
msg.value	uint256	Maximum ETH sold
tokens_bought	uint256	Amount of ERC20 tokens bought
deadline	uint256	Transaction deadline
recipient	address	Address that receives ERC20 tokens

Returns	
uint256	Amount of ETH sold

Smart Contract

```
@payable
ethToTokenTransferOutput(
  tokens_bought: uint256,
  deadline: uint256,
  recipient: address
): uint256
```

Web3

```
exchangeContract.methods
  .ethToTokenTransferOutput(tokens_bought, deadline, (recipient: String))
  .send({ value: ethValue })
```

tokenToEthSwapInput

Parameter	Type	Description
tokens_sold	uint256	Amount of ERC20 tokens sold
min_eth	uint256	Minimum ETH bought
deadline	uint256	Transaction deadline

Returns	
uint256	Amount of ETH bought

Smart Contract

```
tokenToEthSwapInput(  
    tokens_sold: uint256,  
    min_eth: uint256,  
    deadline: uint256  
) : uint256
```

Web3

```
exchangeContract.methods.tokenToEthSwapInput(tokens_sold, min_eth, deadline).send()
```

tokenToEthTransferInput

Parameter	Type	Description
tokens_sold	uint256	Amount of ERC20 tokens sold
min_eth	uint256	Minimum ETH bought
deadline	uint256	Transaction deadline
recipient	address	Address that receives ETH

Returns	
uint256	Amount of ETH bought

Smart Contract

```
tokenToEthTransferInput(  
    tokens_sold: uint256,  
    min_eth: uint256,  
    deadline: uint256,  
    recipient: address  
) : uint256
```

Web3

```
exchangeContract.methods.tokenToEthTransferInput(tokens_sold, min_eth, deadline, recipient).send()
```

tokenToEthSwapOutput

Parameter	Type	Description
eth_bought	uint256	Amount of ETH bought

max_tokens	uint256	Maximum ERC20 tokens sold
deadline	uint256	Transaction deadline

Returns	
uint256	Amount of ERC20 tokens sold

Smart Contract

```
tokenToEthSwapOutput(
    eth_bought: uint256,
    max_tokens: uint256,
    deadline: uint256
): uint256
```

Web3

```
exchangeContract.methods.tokenToEthSwapOutput(eth_bought, max_tokens, (deadline: Integer)).send()
```

tokenToEthTransferOutput

Parameter	Type	Description
eth_bought	uint256	Amount of ETH bought
max_tokens	uint256	Maximum ERC20 tokens sold
deadline	uint256	Transaction deadline
recipient	address	Address that receives ETH

Returns	
uint256	Amount of ERC20 tokens sold

Smart Contract

```
tokenToEthTransferOutput(
    eth_bought: uint256,
    max_tokens: uint256,
    deadline: uint256,
    recipient: address
): uint256
```

Web3

```
exchangeContract.methods
  .tokenToEthTransferOutput(eth_bought, max_tokens, (deadline: Integer), (recipient: String))
  .send()
```

tokenToTokenSwapInput

Parameter	Type	Description
tokens_sold	uint256	Amount of input ERC20 tokens sold
min_tokens_bought	uint256	Minimum output ERC20 tokens bought
min_eth_bought	uint256	Minimum ETH bought as intermediary
deadline	uint256	Transaction deadline

token_addr	address	Address of output ERC20 token
------------	---------	-------------------------------

Returns	
uint256	Amount of output ERC20 tokens bought

Smart Contract

```
tokenToTokenSwapInput(
    tokens_sold: uint256,
    min_tokens_bought: uint256,
    min_eth_bought: uint256,
    deadline: uint256,
    token_addr: address
): uint256
```

Web3

```
exchangeContract.methods
  .tokenToTokenSwapInput(tokens_sold, min_tokens_bought, min_eth_bought, deadline, token_addr)
  .send()
```

tokenToTokenTransferInput

Parameter	Type	Description
tokens_sold	uint256	Amount of input ERC20 tokens sold
min_tokens_bought	uint256	Minimum output ERC20 tokens bought
min_eth_bought	uint256	Minimum ETH bought as intermediary
deadline	uint256	Transaction deadline
recipient	address	Address that receives output ERC20 tokens
token_addr	address	Address of output ERC20 token

Returns	
uint256	Amount of output ERC20 tokens bought

Smart Contract

```
tokenToTokenTransferInput(
    tokens_sold: uint256,
    min_tokens_bought: uint256,
    min_eth_bought: uint256,
    deadline: uint256,
    recipient: address
    token_addr: address
): uint256
```

Web3

```
exchangeContract.methods
  .tokenToTokenTransferInput(tokens_sold, min_tokens_bought, min_eth_bought, deadline, recipient, token_addr)
  .send()
```

tokenToTokenSwapOutput

Parameter	Type	Description
-----------	------	-------------

tokens_bought	uint256	Amount of output ERC20 tokens bought
max_tokens_sold	uint256	Maximum input ERC20 tokens bought
max_eth_sold	uint256	Maximum ETH bought as intermediary
deadline	uint256	Transaction deadline
token_addr	address	Address of output ERC20 token

Returns	
uint256	Amount of input ERC20 tokens sold

Smart Contract

```
tokenToTokenSwapOutput(
    tokens_bought: uint256,
    max_tokens_sold: uint256,
    max_eth_sold: uint256,
    deadline: uint256,
    token_addr: address
): uint256
```

Web3

```
exchangeContract.methods
  .tokenToTokenSwapOutput(tokens_bought, max_tokens_sold, max_eth_sold, deadline, token_addr)
  .send()
```

tokenToTokenTransferOutput

Parameter	Type	Description
tokens_bought	uint256	Amount of output ERC20 tokens bought
max_tokens_sold	uint256	Maximum input ERC20 tokens bought
max_eth_sold	uint256	Maximum ETH bought as intermediary
deadline	uint256	Transaction deadline
recipient	address	Address that receives output ERC20 tokens
token_addr	address	Address of output ERC20 token

Returns	
uint256	Amount of input ERC20 tokens sold

Smart Contract

```
tokenToTokenTransferOutput(
    tokens_bought: uint256,
    max_tokens_sold: uint256,
    max_eth_sold: uint256,
    deadline: uint256,
    recipient: address,
    token_addr: address
): uint256
```

Web3

```

exchangeContract.methods
  .tokenToTokenTransferOutput(tokens_bought, max_tokens_sold, max_eth_sold, deadline, recipient, token_addr)
  .send()

```

tokenToExchangeSwapInput

Parameter	Type	Description
tokens_sold	uint256	Amount of input ERC20 tokens sold
min_tokens_bought	uint256	Minimum output ERC20 tokens bought
min_eth_bought	uint256	Minimum ETH bought as intermediary
deadline	uint256	Transaction deadline
exchange_addr	address	Address of output ERC20 token exchange

Returns	
uint256	Amount of output ERC20 tokens bought

Smart Contract

```

tokenToTokenSwapInput(
  tokens_sold: uint256,
  min_tokens_bought: uint256,
  min_eth_bought: uint256,
  deadline: uint256,
  exchange_addr: address
): uint256

```

Web3

```

exchangeContract.methods
  .tokenToTokenSwapInput(tokens_sold, min_tokens_bought, min_eth_bought, deadline, exchange_addr)
  .send()

```

tokenToExchangeTransferInput

Parameter	Type	Description
tokens_sold	uint256	Amount of input ERC20 tokens sold
min_tokens_bought	uint256	Minimum output ERC20 tokens bought
min_eth_bought	uint256	Minimum ETH bought as intermediary
deadline	uint256	Transaction deadline
recipient	address	Address that receives output ERC20 tokens
exchange_addr	address	Address of output ERC20 token exchange

Returns	
uint256	Amount of output ERC20 tokens bought

Smart Contract

```

tokenToExchangeTransferInput(
  tokens_sold: uint256,
  min_tokens_bought: uint256,
  min_eth_bought: uint256,

```

```

    deadline: uint256,
    recipient: address
    exchange_addr: address
): uint256

```

Web3

```

exchangeContract.methods
.tokenToExchangeTransferInput(tokens_sold, min_tokens_bought, min_eth_bought, deadline, recipient, exchange_addr)
.send()

```

tokenToExchangeSwapOutput

Parameter	Type	Description
tokens_bought	uint256	Amount of output ERC20 tokens bought
max_tokens_sold	uint256	Maximum input ERC20 tokens bought
max_eth_sold	uint256	Maximum ETH bought as intermediary
deadline	uint256	Transaction deadline
exchange_addr	address	Address of output ERC20 token exchange

Returns	
uint256	Amount of input ERC20 tokens sold

Smart Contract

```

tokenToExchangeSwapOutput(
    tokens_bought: uint256,
    max_tokens_sold: uint256,
    max_eth_sold: uint256,
    deadline: uint256,
    exchange_addr: address
): uint256

```

Web3

```

exchangeContract.methods
.tokenToExchangeSwapOutput(tokens_bought, max_tokens_sold, max_eth_sold, deadline, exchange_addr)
.send()

```

tokenToExchangeTransferOutput

Parameter	Type	Description
tokens_bought	uint256	Amount of output ERC20 tokens bought
max_tokens_sold	uint256	Maximum input ERC20 tokens bought
max_eth_sold	uint256	Maximum ETH bought as intermediary
deadline	uint256	Transaction deadline
recipient	address	Address that receives output ERC20 tokens
exchange_addr	address	Address of output ERC20 token exchange

Returns	
uint256	Amount of input ERC20 tokens sold

Smart Contract

```
tokenToExchangeTransferOutput (
    tokens_bought: uint256,
    max_tokens_sold: uint256,
    max_eth_sold: uint256,
    deadline: uint256,
    recipient: address,
    exchange_addr: address
): uint256
```

Web3

```
exchangeContract.methods
  .tokenToExchangeTransferOutput(tokens_bought, max_tokens_sold, max_eth_sold, deadline, recipient, exchange_addr)
  .send()
```

getEthToTokenInputPrice

Parameter	Type	Description
eth_sold	uint256	Amount of ETH sold

Returns	
uint256	Amount of ERC20 tokens that can be bought

Smart Contract

```
@constant
getEthToTokenInputPrice(eth_sold: uint256): uint256
```

Web3

```
exchangeContract.methods.getEthToTokenInputPrice(eth_sold).call()
```

getEthToTokenOutputPrice

Parameter	Type	Description
tokens_bought	uint256	Amount of ERC20 tokens bought

Returns	
uint256	Amount of ETH that must be sold

Smart Contract

```
@constant
getEthToTokenOutputPrice(tokens_bought: uint256): uint256
```

Web3

```
exchangeContract.methods.getEthToTokenOutputPrice(tokens_bought).call()
```

getTokenToEthInputPrice

Parameter	Type	Description
tokens_sold	uint256	Amount of ERC20 tokens sold

Returns	
uint256	Amount of ETH that can be bought

Smart Contract

```
@constant
getTokenToEthInputPrice(tokens_sold: uint256): uint256
```

Web3

```
exchangeContract.methods.getTokenToEthInputPrice(tokens_sold).call()
```

getTokenToEthOutputPrice

Parameter	Type	Description
eth_bought	uint256	Amount of ETH bought

Returns	
uint256	Amount of ERC20 tokens that must be sold

Smart Contract

```
@constant
getTokenToEthOutputPrice(eth_bought: uint256): uint256
```

Web3

```
exchangeContract.methods.getTokenToEthOutputPrice(eth_bought).call()
```

tokenAddress

Returns	
address	Address of ERC20 token sold on exchange

Smart Contract

```
@constant
tokenAddress(): address
```

Web3

```
exchangeContract.methods.tokenAddress().call()
```

factoryAddress

Returns	
address	Address of factory that created exchange

Smart Contract

```
@constant  
factoryAddress(): address
```

Web3

```
exchangeContract.methods.factoryAddress().call()
```

name

Returns	
bytes32	Name of liquidity token

Smart Contract

```
# all exchange contracts have the same name  
@constant  
name(): bytes32 // Uniswap V1
```

Web3

```
exchangeContract.methods.tokenAddress().call()
```

symbol

Returns	
bytes32	Symbol of liquidity token

Smart Contract

```
# all exchange contracts have the same symbol  
@constant  
symbol(): bytes32 // UNI-V1
```

Web3

```
exchangeContract.methods.tokenAddress().call()
```

decimals

Returns	
uint256	Decimals of liquidity token

Smart Contract

```
# all exchange contracts have the same decimals  
@constant  
decimals(): uint256 // 18
```

Web3

```
exchangeContract.methods.decimals().call()
```

balanceOf

Parameter	Type	Description
_owner	address	Ethereum address

Returns	
uint256	Liquidity token balance of address

Smart Contract

```
@constant  
balanceOf(_owner: address): uint256
```

Web3

```
exchangeContract.methods.balanceOf(_owner).call()
```

transfer

Parameter	Type	Description
_to	address	Recipient address
_value	uint256	Amount transferred

Returns	
bool	True if successful. Reverts or false on failure

Smart Contract

```
transfer(  
    _to: address,  
    _value : uint256  
) : bool
```

Web3

```
exchangeContract.methods.transfer(_to, _value).send()
```

transferFrom

Parameter	Type	Description
_from	address	Sender address
_to	address	Recipient address
_value	uint256	Amount transferred

Returns	
bool	True if successful. Reverts or false on failure

Smart Contract

```
transferFrom(  
    _from: address,  
    _to: address,  
    _value : uint256  
) : bool
```

Web3

```
exchangeContract.methods.transferFrom(_from, _to, _value).send()
```

approve

Parameter	Type	Description
_spender	address	Address of approved spender
_value	uint256	Spender allowance

Returns	
bool	True if successful. Reverts or false on failure

Smart Contract

```
approve(  
    _spender: address,  
    _value: uint256  
) : bool
```

Web3

```
exchangeContract.methods.approve(_spender, _value).send()
```

allowance

Parameter	Type	Description
_owner	address	Address of liquidity token owner
_spender	uint256	Address of approved spender

Returns	
uint256	Spender allowance

Smart Contract

```
allowance(  
    _owner: address,  
    _spender: address  
) : uint256
```

Web3

```
exchangeContract.methods.allowance(_owner, _spender).call()
```

id: interfaces title: Interfaces

Factory

Solidity

```
interface UniswapFactoryInterface {
    // Public Variables
    address public exchangeTemplate;
    uint256 public tokenCount;
    // Create Exchange
    function createExchange(address token) external returns (address exchange);
    // Get Exchange and Token Info
    function getExchange(address token) external view returns (address exchange);
    function getToken(address exchange) external view returns (address token);
    function getTokenWithId(uint256 tokenId) external view returns (address token);
    // Never use
    function initializeFactory(address template) external;
}
```

Vyper

```
contract UniswapFactoryInterface():
    # Create Exchange
    def createExchange(token: address) -> address: modifying
    # Public Variables
    def exchangeTemplate() -> address: constant
    def tokenCount() -> uint256: constant
    # Get Exchange and Token Info
    def getExchange(token_addr: address) -> address: constant
    def getToken(exchange: address) -> address: constant
    def getTokenWithId(token_id: uint256) -> address: constant
    # Initialize Factory
    def initializeFactory(template: address): modifying
```

Exchange

Solidity

```
interface UniswapExchangeInterface {
    // Address of ERC20 token sold on this exchange
    function tokenAddress() external view returns (address token);
    // Address of Uniswap Factory
    function factoryAddress() external view returns (address factory);
    // Provide Liquidity
    function addLiquidity(uint256 min_liquidity, uint256 max_tokens, uint256 deadline) external payable returns (uint256);
    function removeLiquidity(uint256 amount, uint256 min_eth, uint256 min_tokens, uint256 deadline) external
    returns (uint256, uint256);
    // Get Prices
    function getEthToTokenInputPrice(uint256 eth_sold) external view returns (uint256 tokens_bought);
    function getEthToTokenOutputPrice(uint256 tokens_bought) external view returns (uint256 eth_sold);
    function getTokenToEthInputPrice(uint256 tokens_sold) external view returns (uint256 eth_bought);
    function getTokenToEthOutputPrice(uint256 eth_bought) external view returns (uint256 tokens_sold);
    // Trade ETH to ERC20
    function ethToTokenSwapInput(uint256 min_tokens, uint256 deadline) external payable returns (uint256
tokens_bought);
    function ethToTokenTransferInput(uint256 min_tokens, uint256 deadline, address recipient) external payable
returns (uint256 tokens_bought);
    function ethToTokenSwapOutput(uint256 tokens_bought, uint256 deadline) external payable returns (uint256
eth_sold);
    function ethToTokenTransferOutput(uint256 tokens_bought, uint256 deadline, address recipient) external payable
returns (uint256 eth_sold);
```

```

// Trade ERC20 to ETH
function tokenToEthSwapInput(uint256 tokens_sold, uint256 min_eth, uint256 deadline) external returns (uint256 eth_bought);
function tokenToEthTransferInput(uint256 tokens_sold, uint256 min_eth, uint256 deadline, address recipient) external returns (uint256 eth_bought);
function tokenToEthSwapOutput(uint256 eth_bought, uint256 max_tokens, uint256 deadline) external returns (uint256 tokens_sold);
function tokenToEthTransferOutput(uint256 eth_bought, uint256 max_tokens, uint256 deadline, address recipient) external returns (uint256 tokens_sold);
// Trade ERC20 to ERC20
function tokenToTokenSwapInput(uint256 tokens_sold, uint256 min_tokens_bought, uint256 min_eth_bought, uint256 deadline, address token_addr) external returns (uint256 tokens_bought);
function tokenToTokenTransferInput(uint256 tokens_sold, uint256 min_tokens_bought, uint256 min_eth_bought, uint256 deadline, address recipient, address token_addr) external returns (uint256 tokens_bought);
function tokenToTokenSwapOutput(uint256 tokens_bought, uint256 max_tokens_sold, uint256 max_eth_sold, uint256 deadline, address token_addr) external returns (uint256 tokens_sold);
function tokenToTokenTransferOutput(uint256 tokens_bought, uint256 max_tokens_sold, uint256 max_eth_sold, uint256 deadline, address recipient, address token_addr) external returns (uint256 tokens_sold);
// Trade ERC20 to Custom Pool
function tokenToExchangeSwapInput(uint256 tokens_sold, uint256 min_tokens_bought, uint256 min_eth_bought, uint256 deadline, address exchange_addr) external returns (uint256 tokens_bought);
function tokenToExchangeTransferInput(uint256 tokens_sold, uint256 min_tokens_bought, uint256 min_eth_bought, uint256 deadline, address recipient, address exchange_addr) external returns (uint256 tokens_bought);
function tokenToExchangeSwapOutput(uint256 tokens_bought, uint256 max_tokens_sold, uint256 max_eth_sold, uint256 deadline, address exchange_addr) external returns (uint256 tokens_sold);
function tokenToExchangeTransferOutput(uint256 tokens_bought, uint256 max_tokens_sold, uint256 max_eth_sold, uint256 deadline, address recipient, address exchange_addr) external returns (uint256 tokens_sold);
// ERC20 compatibility for liquidity tokens
bytes32 public name;
bytes32 public symbol;
uint256 public decimals;
function transfer(address _to, uint256 _value) external returns (bool);
function transferFrom(address _from, address _to, uint256 value) external returns (bool);
function approve(address _spender, uint256 _value) external returns (bool);
function allowance(address _owner, address _spender) external view returns (uint256);
function balanceOf(address _owner) external view returns (uint256);
function totalSupply() external view returns (uint256);
// Never use
function setup(address token_addr) external;
}

```

Vyper

```

contract UniswapExchangeInterface():
    # Public Variables
    def tokenAddress() -> address: constant
    def factoryAddress() -> address: constant
    # Providing Liquidity
    def addLiquidity(min_liquidity: uint256, max_tokens: uint256, deadline: timestamp) -> uint256: modifying
    def removeLiquidity(amount: uint256, min_eth: uint256(wei), min_tokens: uint256, deadline: timestamp) -> (uint256(wei), uint256): modifying
    # Trading
    def ethToTokenSwapInput(min_tokens: uint256, deadline: timestamp) -> uint256: modifying
    def ethToTokenTransferInput(min_tokens: uint256, deadline: timestamp, recipient: address) -> uint256: modifying
    def ethToTokenSwapOutput(tokens_bought: uint256, deadline: timestamp) -> uint256(wei): modifying
    def ethToTokenTransferOutput(tokens_bought: uint256, deadline: timestamp, recipient: address) -> uint256(wei): modifying
    def tokenToEthSwapInput(tokens_sold: uint256, min_eth: uint256(wei), deadline: timestamp) -> uint256(wei): modifying
    def tokenToEthTransferInput(tokens_sold: uint256, min_eth: uint256(wei), deadline: timestamp, recipient: address) -> uint256(wei): modifying
    def tokenToEthSwapOutput(eth_bought: uint256(wei), max_tokens: uint256, deadline: timestamp) -> uint256: modifying
    def tokenToEthTransferOutput(eth_bought: uint256(wei), max_tokens: uint256, deadline: timestamp, recipient: address) -> uint256: modifying
    def tokenToTokenSwapInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_bought: uint256(wei), deadline: timestamp, token_addr: address) -> uint256: modifying
    def tokenToTokenTransferInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_bought: uint256(wei), deadline: timestamp, recipient: address, token_addr: address) -> uint256: modifying

```

```

def tokenToTokenSwapOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_sold: uint256(wei),
deadline: timestamp, token_addr: address) -> uint256: modifying
    def tokenToTokenTransferOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_sold: uint256(wei),
deadline: timestamp, recipient: address, token_addr: address) -> uint256: modifying
        def tokenToExchangeSwapInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_bought: uint256(wei),
deadline: timestamp, exchange_addr: address) -> uint256: modifying
            def tokenToExchangeTransferInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_bought:
uint256(wei), deadline: timestamp, recipient: address, exchange_addr: address) -> uint256: modifying
                def tokenToExchangeSwapOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_sold: uint256(wei),
deadline: timestamp, exchange_addr: address) -> uint256: modifying
                    def tokenToExchangeTransferOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_sold: uint256(wei),
deadline: timestamp, recipient: address, exchange_addr: address) -> uint256: modifying
                        # Get Price
                        def getEthToTokenInputPrice(eth_sold: uint256(wei)) -> uint256: constant
                        def getEthToTokenOutputPrice(tokens_bought: uint256) -> uint256(wei): constant
                        def getTokenToEthInputPrice(tokens_sold: uint256) -> uint256(wei): constant
                        def getTokenToEthOutputPrice(eth_bought: uint256(wei)) -> uint256: constant
                        # Pool Token ERC20 Compatibility
                        def balanceOf() -> address: constant
                        def allowance(_owner : address, _spender : address) -> uint256: constant
                        def transfer(_to : address, _value : uint256) -> bool: modifying
                        def transferFrom(_from : address, _to : address, _value : uint256) -> bool: modifying
                        def approve(_spender : address, _value : uint256) -> bool: modifying
                        # Setup
                        def setup(token_addr: address): modifying

```

ERC20 Token

Solidity

```

interface ERC20Interface {
    function totalSupply() public view returns (uint);
    function balanceOf(address tokenOwner) public view returns (uint balance);
    function allowance(address tokenOwner, address spender) public view returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);
    // optional
    function name() external view returns (string);
    function symbol() external view returns (string);
    function decimals() external view returns (string);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}

```

Vyper

```

contract ERC20Interface():
    def totalSupply() -> uint256: constant
    def balanceOf(_owner: address) -> uint256: constant
    def allowance(_owner : address, _spender : address) -> uint256: constant
    def transfer(_to : address, _value : uint256) -> bool: modifying
    def approve(_spender : address, _value : uint256) -> bool: modifying
    def transferFrom(_from : address, _to : address, _value : uint256) -> bool: modifying
    # optional
    def name() -> bytes32: constant
    def symbol() -> bytes32: constant
    def decimals() -> uint256: constant

```

id: how-uniswap-works title: How Uniswap works

Uniswap is an *automated liquidity protocol* powered by a [constant product formula](#) and implemented in a system of non-upgradeable smart contracts on the [Ethereum](#) blockchain. It obviates the need for trusted intermediaries, prioritizing **decentralization, censorship resistance, and security**. Uniswap is **open-source software** licensed under the [GPL](#).

Each Uniswap smart contract, or pair, manages a liquidity pool made up of reserves of two [ERC-20](#) tokens.

Anyone can become a liquidity provider (LP) for a pool by depositing an equivalent value of each underlying token in return for pool tokens. These tokens track pro-rata LP shares of the total reserves, and can be redeemed for the underlying assets at any time.

Pairs act as automated market makers, standing ready to accept one token for the other as long as the “constant product” formula is preserved. This formula, most simply expressed as $x * y = k$, states that trades must not change the product (k) of a pair’s reserve balances (x and y). Because k remains unchanged from the reference frame of a trade, it is often referred to as the invariant. This formula has the desirable property that larger trades (relative to reserves) execute at exponentially worse rates than smaller ones.

In practice, Uniswap applies a 0.30% fee to trades, which is added to reserves. As a result, each trade actually increases k . This functions as a payout to LPs, which is realized when they burn their pool tokens to withdraw their portion of total reserves. In the future, this fee may be reduced to 0.25%, with the remaining 0.05% withheld as a protocol-wide charge.

Because the relative price of the two pair assets can only be changed through trading, divergences between the Uniswap price and external prices create arbitrage opportunities. This mechanism ensures that Uniswap prices always trend toward the market-clearing price.

Further reading

To see how token swaps work in practice, and to walk through the lifecycle of a swap, check out [Swaps](#). Or, to see how liquidity pools work, see [Pools](#).

Ultimately, of course, the Uniswap protocol is just smart contract code running on Ethereum. To understand how they work, head over to [Smart Contracts](#).

id: ecosystem-participants title: Ecosystem Participants

The Uniswap ecosystem is primarily comprised of three types of users: liquidity providers, traders, and developers. Liquidity providers are incentivized to contribute [ERC-20](#) tokens to common liquidity pools. Traders can swap these tokens for one another for a fixed [0.30% fee](#) (which goes to liquidity providers). Developers can integrate directly with Uniswap smart contracts to power new and exciting interactions with tokens, trading interfaces, retail experiences, and more.

In total, interactions between these classes create a positive feedback loop, fueling digital economies by defining a common language through which tokens can be pooled, traded and used.

Liquidity Providers

Liquidity providers, or LPs, are not a homogenous group:

- Passive LPs are token holders who wish to passively invest their assets to accumulate trading fees.
- Professional LPs are focused on market making as their primary strategy. They usually develop custom tools and ways of tracking their liquidity positions across different DeFi projects.
- Token projects sometimes choose to become LPs to create a liquid marketplace for their token. This allows tokens to be bought and sold more easily, and unlocks interoperability with other DeFi projects through Uniswap.
- Finally, some DeFi pioneers are exploring complex liquidity provision interactions like incentivized liquidity, liquidity as collateral, and other experimental strategies. Uniswap is the perfect protocol for projects to experiment with these kinds of ideas.

Traders

There are several categories of traders in the protocol ecosystem:

- Speculators use a variety of community built tools and products to swap tokens using liquidity pulled from the Uniswap protocol.
- Arbitrage bots seek profits by comparing prices across different platforms to find an edge. (Though it might seem extractive, these bots actually help equalize prices across broader Ethereum markets and keep things fair.)

- DAPP users buy tokens on Uniswap for use in other applications on Ethereum.
- Smart contracts that execute trades on the protocol by implementing swap functionality (from products like DEX aggregators to custom Solidity scripts).

In all cases, trades are subject to the same flat fee for trading on the protocol. Each is important for increasing the accuracy of prices and incentivizing liquidity.

Developers/Projects

There are far too many ways Uniswap is used in the wider Ethereum ecosystem to count, but some examples include:

- The open-source, accessible nature of Uniswap means there are countless UX experiments and front-ends built to offer access to Uniswap functionality. You can find Uniswap functions in most of the major DeFi dashboard projects. There are also many [Uniswap-specific tools](#) built by the community.
- Wallets often integrate swapping and liquidity provision functionality as a core offering of their product.
- DEX (decentralized exchange) aggregators pull liquidity from many liquidity protocols to offer traders the best prices by splitting their trades. Uniswap is the biggest single decentralized liquidity source for these projects.
- Smart contract developers use the suite of functions available to invent new DeFi tools and other various experimental ideas. See projects like [Unisocks](#) or [Zora](#), among many, many others.

Uniswap Team and Community

The Uniswap team along with the broader Uniswap community drives development of the protocol and ecosystem.

id: smart-contracts title: Smart contracts

Uniswap V2 is a binary smart contract system. [Core](#) contracts provide fundamental safety guarantees for all parties interacting with Uniswap. [Periphery](#) contracts interact with one or more core contracts but are not themselves part of the core.

Core

[Source code](#)

The core consists of a singleton [factory](#) and many [pairs](#), which the factory is responsible for creating and indexing. These contracts are quite minimal, even brutalist. The simple rationale for this is that contracts with a smaller surface area are easier to reason about, less bug-prone, and more functionally elegant. Perhaps the biggest upside of this design is that many desired properties of the system can be asserted directly in the code, leaving little room for error. One downside, however, is that core contracts are somewhat user-unfriendly. In fact, interacting directly with these contracts is not recommended for most use cases. Instead, a periphery contract should be used.

Factory

[Reference documentation](#)

The factory holds the generic bytecode responsible for powering pairs. Its primary job is to create one and only one smart contract per unique token pair. It also contains logic to turn on the protocol charge.

Pairs

[Reference documentation](#)

[Reference documentation \(ERC-20\)](#)

Pairs have two primary purposes: serving as automated market makers and keeping track of pool token balances. They also expose data which can be used to build decentralized price oracles.

Periphery

[Source code](#)

The periphery is a constellation of smart contracts designed to support domain-specific interactions with the core. Because of Uniswap's permissionless nature, the contracts described below have no special privileges, and are in fact only a small subset of the universe of possible periphery-like contracts. However, they are useful examples of how to safely and efficiently interact with Uniswap V2.

Library

[Reference documentation](#)

The library provides a variety of convenience functions for fetching data and pricing.

Router

[Reference documentation](#)

The router, which uses the library, fully supports all the basic requirements of a front-end offering trading and liquidity management functionality. Notably, it natively supports multi-pair trades (e.g. x to y to z), treats ETH as a first-class citizen, and offers meta-transactions for removing liquidity.

Design Decisions

The following sections describe some of the notable design decisions made in Uniswap V2. These are safe to skip unless you're interested in gaining a deep technical understanding of how V2 works under the hood, or writing smart contract integrations!

Sending Tokens

Typically, smart contracts which need tokens to perform some functionality require would-be interactors to first make an approval on the token contract, then call a function that in turn calls transferFrom on the token contract. This is *not* how V2 pairs accept tokens. Instead, pairs check their token balances at the *end* of every interaction. Then, at the beginning of the *next* interaction, current balances are differenced against the stored values to determine the amount of tokens that were sent by the current interactor. See the [whitepaper](#) for a justification of why this is the case, but the takeaway is that **tokens must be transferred to the pair before calling any token-requiring method** (the one exception to this rule is [Flash Swaps](#)).

WETH

Unlike Uniswap V1 pools, V2 pairs do not support ETH directly, so $\text{ETH} \rightleftharpoons \text{ERC-20}$ pairs must be emulated with WETH. The motivation behind this choice was to remove ETH-specific code in the core, resulting in a leaner codebase. End users can be kept fully ignorant of this implementation detail, however, by simply wrapping/unwrapping ETH in the periphery.

The router fully supports interacting with any WETH pair via ETH.

Minimum Liquidity

To ameliorate rounding errors and increase the theoretical minimum tick size for liquidity provision, pairs burn the first [MINIMUM LIQUIDITY](#) pool tokens. For the vast majority of pairs, this will represent a trivial value. The burning happens automatically during the first liquidity provision, after which point the [totalSupply](#) is forevermore bounded.

id: glossary title: Glossary

Automated market maker

An automated market maker is a smart contract on Ethereum that holds on-chain liquidity reserves. Users can trade against these reserves at prices set by an automated market making formula.

Constant product formula

The automated market making algorithm used by Uniswap. See [x*y=k](#).

ERC20

ERC20 tokens are fungible tokens on Ethereum. Uniswap supports all standard ERC20 implementations.

Factory

A smart contract that deploys a unique smart contract for any ERC20/ERC20 trading pair.

Pair

A smart contract deployed from the Uniswap V2 Factory that enables trading between two ERC20 tokens.

Pool

Liquidity within a pair is pooled across all liquidity providers.

Liquidity provider / LP

A liquidity provider is someone who deposits an equivalent value of two ERC20 tokens into the liquidity pool within a pair. Liquidity providers take on price risk and are compensated with fees.

Mid price

The price between what users can buy and sell tokens at a given moment. In Uniswap this is the ratio of the two ERC20 token reserves.

Price impact

The difference between the mid-price and the execution price of a trade.

Slippage

The amount the price moves in a trading pair between when a transaction is submitted and when it is executed.

Core

Smart contracts that are essential for Uniswap to exist. Upgrading to a new version of core would require a liquidity migration.

Periphery

External smart contracts that are useful, but not required for Uniswap to exist. New periphery contracts can always be deployed without migrating liquidity.

Flash swap

A trade that uses the tokens being purchased before paying for them.

`x * y = k`

The constant product formula.

Invariant

The "k" value in the constant product formula

id: swaps title: Swaps subtitle: Learn about the core functionality of the uniswap protocol. Token Swaps.

Introduction

Token swaps in Uniswap are a simple way to trade one ERC-20 token for another.

For end-users, swapping is intuitive: a user picks an input token and an output token. They specify an input amount, and the protocol calculates how much of the output token they'll receive. They then execute the swap with one click, receiving the output token in their wallet immediately.

In this guide, we'll look at what happens during a swap at the protocol level in order to gain a deeper understanding of how Uniswap works.

Swaps in Uniswap are different from trades on traditional platforms. Uniswap does not use an order book to represent liquidity or determine prices. Uniswap uses an automated market maker mechanism to provide instant feedback on rates and slippage.

As we learned in [Protocol Overview](#), each pair on Uniswap is actually underpinned by a liquidity pool. Liquidity pools are smart contracts that hold balances of two unique tokens and enforces rules around depositing and withdrawing them.

This rule is the [constant product formula](#). When either token is withdrawn (purchased), a proportional amount of the other must be deposited (sold), in order to maintain the constant.

Anatomy of a swap

At the most basic level, all swaps in Uniswap V2 happen within a single function, aptly named `swap`:

```
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data);
```

Receiving tokens

As is probably clear from the function signature, Uniswap requires `swap` callers to specify how many output tokens they would like to receive via the `amount{0,1}Out` parameters, which correspond to the desired amount of `token{0,1}`.

Sending Tokens

What's not as clear is how Uniswap receives tokens as payment for the swap. Typically, smart contracts which need tokens to perform some functionality require callers to first make an approval on the token contract, then call a function that in turn calls transferFrom on the token contract. This is *not* how V2 pairs accept tokens. Instead, pairs check their token balances at the *end* of every interaction. Then, at the beginning of the next interaction, current balances are differenced against the stored values to determine the amount of tokens that were sent by the current interactor. See the [whitepaper](#) for a justification of why this is the case.

The takeaway is that **tokens must be transferred to pairs before swap is called** (the one exception to this rule is [Flash Swaps](#)). This means that to safely use the `swap` function, it must be called from *another smart contract*. The alternative (transferring tokens to the pair and then calling `swap`) is not safe to do non-atomically because the sent tokens would be vulnerable to arbitrage.

Developer resources

- To see how to implement token swaps in a smart contract read [Trading from a smart contract](#).
 - To see how to execute a swap from an interface read [Trading \(SDK\)](#)
-

id: pools title: Pools

Introduction

Each Uniswap liquidity pool is a trading venue for a pair of ERC20 tokens. When a pool contract is created, its balances of each token are 0; in order for the pool to begin facilitating trades, someone must seed it with an initial deposit of each token. This first liquidity provider is the one who sets the initial price of the pool. They are incentivized to deposit an equal *value* of both tokens into the pool. To see why, consider the case where the first liquidity provider deposits tokens at a ratio different from the current market rate. This immediately creates a profitable arbitrage opportunity, which is likely to be taken by an external party.

When other liquidity providers add to an existing pool, they must deposit pair tokens proportional to the current price. If they don't, the liquidity they added is at risk of being arbitrated as well. If they believe the current price is not correct, they may arbitrage it to the level they desire, and add liquidity at that price.

Pool tokens

Whenever liquidity is deposited into a pool, unique tokens known as *liquidity tokens* are minted and sent to the provider's address. These tokens represent a given liquidity provider's contribution to a pool. The proportion of the pool's liquidity provided determines the number of liquidity tokens the provider receives. If the provider is minting a new pool, the number of liquidity tokens they will receive will equal $\text{sqrt}(x * y)$, where x and y represent the amount of each token provided.

Whenever a trade occurs, a 0.3% fee is charged to the transaction sender. This fee is distributed *pro-rata* to all LPs in the pool upon completion of the trade.

To retrieve the underlying liquidity, plus any fees accrued, liquidity providers must "burn" their liquidity tokens, effectively exchanging them for their portion of the liquidity pool, plus the proportional fee allocation.

As liquidity tokens are themselves tradable assets, liquidity providers may sell, transfer, or otherwise use their liquidity tokens in any way they see fit.

[Learn more with advanced topics:](#)

- [Understanding Returns](#)
- [Fees](#)

Why pools?

Uniswap is unique in that it doesn't use an order book to derive the price of an asset or to match buyers and sellers of tokens. Instead, Uniswap uses what are called Liquidity Pools.

Liquidity is typically represented by discrete orders placed by individuals onto a centrally operated order book. A participant looking to provide liquidity or make markets must actively manage their orders, continuously updating them in response to the activity of others in the marketplace.

While order books are foundational to finance and work great for certain usecases, they suffer from a few important limitations that are especially magnified when applied to a decentralized or blockchain-native setting. Order books require intermediary infrastructure to host the orderbook and match orders. This creates points of control and adds additional layers of complexity. They also require active participation and management from

market makers who usually use sophisticated infrastructure and algorithms, limiting participation to advanced traders. Order books were invented in a world with relatively few assets being traded, so it is not surprising they aren't ideal for an ecosystem where anyone can create their own token, and those tokens usually have low liquidity. In sum, with the infrastructural trade-offs presented by a platform like Ethereum, order books are not the native architecture for implementing a liquidity protocol on a blockchain.

Uniswap focuses on the strengths of Ethereum to reimagine token swaps from first principles.

A blockchain-native liquidity protocol should take advantage of the trusted code execution environment, the autonomous and perpetually running virtual machine, and an open, permissionless, and inclusive access model that produces an exponentially growing ecosystem of virtual assets.

It is important to reiterate that a Pool is just a smart contract, operated by users calling functions on it. Swapping tokens is calling `swap` on a Pool contract instance, while providing liquidity is calling `deposit`.

Just how end-users can interact with the Uniswap protocol through the Interface (which in turn interacts with the underlying contracts), developers can interact directly with the smart contracts and integrate Uniswap functionality into their own applications without relying on intermediaries or needing permission.

Developer resources

- To see how to pool tokens in a smart contract read [Providing Liquidity](#).

id: flash-swaps title: Flash Swaps

Uniswap flash swaps allow you to withdraw up to the full reserves of any ERC20 token on Uniswap and execute arbitrary logic at no upfront cost, provided that by the end of the transaction you either:

- pay for the withdrawn ERC20 tokens with the corresponding pair tokens
- return the withdrawn ERC20 tokens along with a small fee

Flash swaps are incredibly useful because they obviate upfront capital requirements and unnecessary order-of-operations constraints for multi-step transactions involving Uniswap.

Examples

Capital Free Arbitrage

One particularly interesting use case for flash swaps is capital-free arbitrage. It's well-known that an integral part of Uniswap's design is to create incentives for arbitrageurs to trade the Uniswap price to a "fair" market price. While game-theoretically sound, this strategy is accessible only to those with sufficient capital to take advantage of arbitrage opportunities. Flash swaps remove this barrier entirely, effectively democratizing arbitrage.

Imagine a scenario where the cost of buying 1 ETH on Uniswap is 200 DAI (which is calculated by calling `getAmountIn` with 1 ETH specified as an exact output), and on Oasis (or any other trading venue), 1 ETH buys 220 DAI. To anyone with 200 DAI available, this situation represents a risk-free profit of 20 DAI. Unfortunately, you may not have 200 DAI lying around. With flash swaps, however, this risk-free profit is available for anyone to take as long as they're able to pay gas fees.

Withdrawing ETH from Uniswap

The first step is to *optimistically* withdraw 1 ETH from Uniswap via a flash swap. This will serve as the capital that we use to execute our arbitrage. Note that in this scenario, we're assuming that:

- 1 ETH is the pre-calculated profit-maximizing trade
- The price has not changed on Uniswap or Oasis since our calculation

It may be the case that we'd like to calculate the profit-maximizing trade on-chain at the moment of execution, which is robust to price movements. This can be somewhat complex, depending on the strategy being executed. However, one common strategy is trading as profitably as possible *against a fixed external price*. (This price may be e.g., the average execution price of one or more orders on Oasis.) If the Uniswap market price is far enough above or below this external price, the following example contains code that calculates the amount to trade over Uniswap for maximum profit: [ExampleSwapToPrice.sol](#).

Trade at External Venue

Once we've obtained our temporary capital of 1 ETH from Uniswap, we now can trade this for 220 DAI on Oasis. Once we've received the DAI, we need to pay Uniswap back. We've mentioned that the amount required to cover 1 ETH is 200 DAI, calculated via `getAmountIn`. So, after sending 200 of the DAI back to the Uniswap pair, you're left with 20 DAI of profit!

Instant Leverage

Flash swaps can be used to improve the efficiency of levering up using lending protocols and Uniswap.

Consider Maker in its simplest form: a system which accepts ETH as collateral and allows DAI to be minted against it while ensuring that the value of the ETH never drops below 150% of the value of the DAI.

Say we use this system to deposit a principal amount of 3 ETH, and mint the maximum amount of DAI. At a price of 1 ETH / 200 DAI, we receive 400 DAI. In theory, we could lever this position up by selling the DAI for more ETH, depositing this ETH, minting the maximum amount of DAI (which would be less this time), and repeating until we've reached our desired leverage level.

It's quite simple to use Uniswap as a liquidity source for the DAI-to-ETH component of this process. However, looping through protocols in this way isn't particularly elegant, and can be gas-intensive.

Luckily, flash swaps enable us to withdraw the *full* ETH amount upfront. If we wanted 2x leverage against our 3 ETH principal, we could simply request 3 ETH in a flash swap and deposit 6 ETH into Maker. This gives us the ability to mint 800 DAI. If we mint as much as we need to cover our flash swap (say 605), the remainder serves as a safety margin against price movements.

Developer resources

- To see how to integrate a flash swap in your smart contract read [Using Flash Swaps](#).
-

id: oracles title: Oracles

Introduction

A price oracle is any tool used to view price information about a given asset. When you look at stock prices on your phone, you are using your phone as a price oracle. Similarly, the app on your phone relies on devices to retrieve price information - likely several, which are aggregated and then displayed to you, the end-user. These are price oracles as well.

When building smart contracts that integrate with DeFi protocols, developers will inevitably run into the price oracle problem. What is the best way to retrieve the price of a given asset on-chain?

Many oracle designs on Ethereum have been implemented on an ad-hoc basis, with varying degrees of decentralization and security. Because of this, the ecosystem has witnessed numerous high-profile hacks where the oracle implementation is the primary attack vector. Some of these vulnerabilities are discussed [here](#).

While there is no one size fits all solution, Uniswap V2 enables developers to build highly decentralized and manipulation-resistant on-chain price oracles, which may solve many of the demands necessary for building robust protocols.

Uniswap V2 solution

Uniswap V2 includes several improvements for supporting manipulation-resistant public price feeds. First, every pair measures (but does not store) the market price at the beginning of each block, before any trades take place. This price is expensive to manipulate because it is set by the last transaction, whether it is a mint, swap, or burn, in a previous block.

To set the measured price to one that is out of sync with the global market price, an attacker has to make a bad trade at the end of a previous block, typically with no guarantee that they will arbitrage it back in the next block. Attackers will lose money to arbitrageurs unless they can "selfishly" mine two blocks in a row. This type of attack presents several challenges and [has not been observed to date](#).

Unfortunately, this alone is not enough. If significant value settles based on the price resulting from this mechanism, an attack's profit will likely outweigh the loss.

Instead, Uniswap V2 adds this end-of-block price to a single cumulative-price variable in the core contract weighted by the amount of time this price existed. **This variable represents a sum of the Uniswap price for every second in the entire history of the contract.**

This variable can be used by external contracts to track accurate time-weighted average prices (TWAPs) across any time interval.

The TWAP is constructed by reading the cumulative price from an ERC20 token pair at the beginning and at the end of the desired interval. The difference in this cumulative price can then be divided by the length of the interval to create a TWAP for that period.

TWAPs can be used directly or as the basis for moving averages (EMAs and SMAs) as needed.

A few notes:

- For a 10-minute TWAP, sample once every 10 minutes. For a 1-week TWAP, sample once every week.
- For a simple TWAP, the cost of manipulation increases (approx. linear) with liquidity on Uniswap, as well as (approx. linear) with the length of time over which you average.
- The Cost of an attack is relatively simple to estimate. Moving the price 5% on a 1-hour TWAP is approximately equal to the amount lost to arbitrage and fees for moving the price 5% every block for 1 hour.

There are some nuances that are good to be aware of when using Uniswap V2 as an oracle, especially where manipulation resistance is concerned. The [whitepaper](#) elaborates on some of them. Additional oracle-focused developer guides and documentation will be released soon.

In the meantime, check out our [example implementation](#) of a 24 hr TWAP Oracle built on Uniswap V2!

Manipulation resistance

The cost of manipulating the price for a specific time period can be roughly estimated as the amount lost to arbitrage and fees every block for the entire period. For larger liquidity pools and over longer time periods, this attack is impractical, as the cost of manipulation typically exceeds the value at stake.

Other factors, such as network congestion, can reduce the cost of attack. For a more in-depth review of the security of Uniswap V2 price oracles, read the [security audit section on Oracle Integrity](#).

Building an oracle

To learn more about building oracles check out [building an Oracle](#) in the developer guides.

id: fees title: Fees

Liquidity provider fees

There is a 0.3% fee for swapping tokens. **This fee is split by liquidity providers proportional to their contribution to liquidity reserves.**

Swapping fees are immediately deposited into liquidity reserves. This increases the value of liquidity tokens, functioning as a payout to all liquidity providers proportional to their share of the pool. Fees are collected by burning liquidity tokens to remove a proportional share of the underlying reserves.

Since fees are added to liquidity pools, the invariant increases at the end of every trade. Within a single transaction, the invariant represents `token0_pool / token1_pool` at the end of the previous transaction.

There are many community-developed tools to determine returns. You can also read more in the docs about how to think about [LP returns](#).

Protocol Fees

At the moment there are no protocol fees. However, it is possible for a 0.05% fee to be turned on in the future.

More information about a potential future protocol fee can be found [here](#).

Protocol Charge Calculation

In the future, it is possible that a protocol-wide charge of 0.05% per trade will take effect. This represents 1/6th (16.6%) of the 0.30% fee. The fee is in effect if `feeTo` is not `address(0)` (`0x000`), indicating that `feeTo` is the recipient of the charge.

This amount would not affect the fee paid by traders, but would affect the amount received by liquidity providers.

Rather than calculating this charge on swaps, which would significantly increase gas costs for all users, the charge is instead calculated when liquidity is added or removed. See the [whitepaper](#) for more details.

id: pricing title: Pricing

How are prices determined?

As we learned in [Protocol Overview](#), each pair on Uniswap is actually underpinned by a liquidity pool. Liquidity pools are smart contracts that hold balances of two unique tokens and enforces rules around depositing and withdrawing them. The primary rule is the [constant product formula](#). When a token is withdrawn (bought), a proportional amount must be deposited (sold) to maintain the constant. The ratio of tokens in the pool, in combination with the constant product formula, ultimately determine the price that a swap executes at.

How Uniswap handles prices

In Uniswap V1, trades are always executed at the "best possible" price, calculated at execution time. Somewhat confusingly, this calculation is actually accomplished with one of two different formulas, depending on whether the trade specifies an exact *input* or *output* amount. Functionally, the difference between these two functions is minuscule, but the very existence of a difference increases conceptual complexity. Initial attempts to support both functions in V2 proved inelegant, and the decision was made to **not provide any pricing functions in the core**. Instead, pairs directly check whether the invariant was satisfied (accounting for fees) after every trade. This means that rather than relying on a pricing function

to also enforce the invariant, V2 pairs simply and transparently ensure their own safety, a nice separation of concerns. One downstream benefit is that V2 pairs will more naturally support other flavors of trades which may emerge, (e.g. trading to a specific price at execution time).

At a high level, in Uniswap V2, *trades must be priced in the periphery*. The good news is that the [library](#) provides a variety of functions designed to make this quite simple, and all swapping functions in the [router](#) are designed with this in mind.

Pricing Trades

When swapping tokens on Uniswap, it's common to want to receive as many output tokens as possible for an *exact input amount*, or to pay as few input tokens as possible for an *exact output amount*. In order to calculate these amounts, a contract must look up the *current reserves* of a pair, in order to understand what the current price is. However, it is *not safe to perform this lookup and rely on the results without access to an external price*.

Say a smart contract naively wants to send 10 DAI to the DAI/WETH pair and receive as much WETH as it can get, given the current reserve ratio. If, when called, the naive smart contract simply looks up the current price and executes the trade, it is *vulnerable to front-running and will likely suffer an economic loss*. To see why, consider a malicious actor who sees this transaction before it is confirmed. They could execute a swap which dramatically changes the DAI/WETH price immediately before the naive swap goes through, wait for the naive swap to execute at a bad rate, and then swap to change the price back to what it was before the naive swap. This attack is fairly cheap and low-risk, and can typically be performed for a profit.

To prevent these types of attacks, it's vital to submit swaps *that have access to knowledge about the "fair" price their swap should execute at*. In other words, swaps need access to an *oracle*, to be sure that the best execution they can get from Uniswap is close enough to what the oracle considers the "true" price. While this may sound complicated, the oracle can be as simple as an *off-chain observation of the current market price of a pair*. Because of arbitrage, it's typically the case that the ratio of the intra-block reserves of a pair is close to the "true" market price. So, if a user submits a trade with this knowledge in mind, they can ensure that the losses due to front-running are tightly bounded. This is how, for example, the Uniswap frontend ensures trade safety. It calculates the optimal input/output amounts given observed intra-block prices, and uses the router to perform the swap, which guarantees the swap will execute at a rate no less than \times % worse than the observed intra-block rate, where \times is a user-specified slippage tolerance (0.5% by default).

There are, of course, other options for oracles, including [native V2 oracles](#).

Exact Input

If you'd like to send an exact amount of input tokens in exchange for as many output tokens as possible, you'll want to use [getAmountsOut](#). The equivalent SDK function is [getOutputAmount](#), or [minimumAmountOut](#) for slippage calculations.

Exact Output

If you'd like to receive an exact amount of output tokens for as few input tokens as possible, you'll want to use [getAmountsIn](#). The equivalent SDK function is [getInputAmount](#), or [maximumAmountIn](#) for slippage calculations.

Swap to Price

For this more advanced use case, see [ExampleSwapToPrice.sol](#).

id: understanding-returns title: Understanding Returns

Uniswap incentivizes users to add liquidity to trading pools by rewarding providers with the fees generated when other users trade with those pools. Market making, in general, is a complex activity. There is a risk of losing money during large and sustained movement in the underlying asset price compared to simply holding an asset.

Risks

To understand the risks associated with providing liquidity you can read <https://medium.com/@pintail/uniswap-a-good-deal-for-liquidity-providers-104c0b6816f2> to get an in-depth look at how to conceptualize a liquidity position.

Example from the article

Consider the case where a liquidity provider adds 10,000 DAI and 100 WETH to a pool (for a total value of \$20,000), the liquidity pool is now 100,000 DAI and 1,000 ETH in total. Because the amount supplied is equal to 10% of the total liquidity, the contract mints and sends the market maker "liquidity tokens" which entitle them to 10% of the liquidity available in the pool. These are not speculative tokens to be traded. They are merely an accounting or bookkeeping tool to keep track of how much the liquidity providers are owed. If others subsequently add/withdraw coins, new liquidity tokens are minted/burned such that everyone's relative percentage share of the liquidity pool remains the same.

Now let's assume the price trades on Coinbase from \$100 to \$150. The Uniswap contract should reflect this change as well after some arbitrage. Traders will add DAI and remove ETH until the new ratio is now 150:1.

*What happens to the liquidity provider? The contract reflects something closer to 122,400 DAI and 817 ETH (to check these numbers are accurate, $122,400 * 817 = 100,000,000$ (our constant product) and $122,400 / 817 = 150$, our new price). Withdrawing the 10% that we are entitled to would now yield 12,240 DAI and 81.7 ETH. The total market value here is \$24,500. Roughly \$500 worth of profit was missed out on as a result of the market making.*

Obviously no one wants to provide liquidity out of charitable means, and the revenue isn't dependent on the ability to flip out of good trades (there is no flipping). Instead, 0.3% of all trade volume is distributed proportionally to all liquidity providers. By default, these fees are put back into the liquidity pool, but can be collected any time. It's difficult to know what the trade-off is between revenues from fees and losses from directional movements without knowing the amount of in-between trades. The more chop and back and forth, the better.

Why is my liquidity worth less than I put in?

To understand why the value of a liquidity provider's stake can go down despite income from fees, we need to look a bit more closely at the formula used by Uniswap to govern trading. The formula really is very simple. If we neglect trading fees, we have the following:

- `eth_liquidity_pool * token_liquidity_pool = constant_product`

In other words, the number of tokens a trader receives for their ETH and vice versa is calculated such that after the trade, the product of the two liquidity pools is the same as it was before the trade. The consequence of this formula is that for trades which are very small in value compared to the size of the liquidity pool we have:

- `eth_price = token_liquidity_pool / eth_liquidity_pool`

Combining these two equations, we can work out the size of each liquidity pool at any given price, assuming constant total liquidity:

- `eth_liquidity_pool = sqrt(constant_product / eth_price)`
- `token_liquidity_pool = sqrt(constant_product * eth_price)`

So let's look at the impact of a price change on a liquidity provider. To keep things simple, let's imagine our liquidity provider supplies 1 ETH and 100 DAI to the Uniswap DAI exchange, giving them 1% of a liquidity pool which contains 100 ETH and 10,000 DAI. This implies a price of 1 ETH = 100 DAI. Still neglecting fees, let's imagine that after some trading, the price has changed; 1 ETH is now worth 120 DAI. What is the new value of the liquidity provider's stake? Plugging the numbers into the formulae above, we have:

- `eth_liquidity_pool = 91.2871`
- `dai_liquidity_pool = 10954.4511`

"Since our liquidity provider has 1% of the liquidity tokens, this means they can now claim 0.9129 ETH and 109.54 DAI from the liquidity pool. But since DAI is approximately equivalent to USD, we might prefer to convert the entire amount into DAI to understand the overall impact of the price change. At the current price then, our liquidity is worth a total of 219.09 DAI. What if the liquidity provider had just held onto their original 1 ETH and 100 DAI? Well, now we can easily see that, at the new price, the total value would be 220 DAI. So our liquidity provider lost out by 0.91 DAI by providing liquidity to Uniswap instead of just holding onto their initial ETH and DAI."

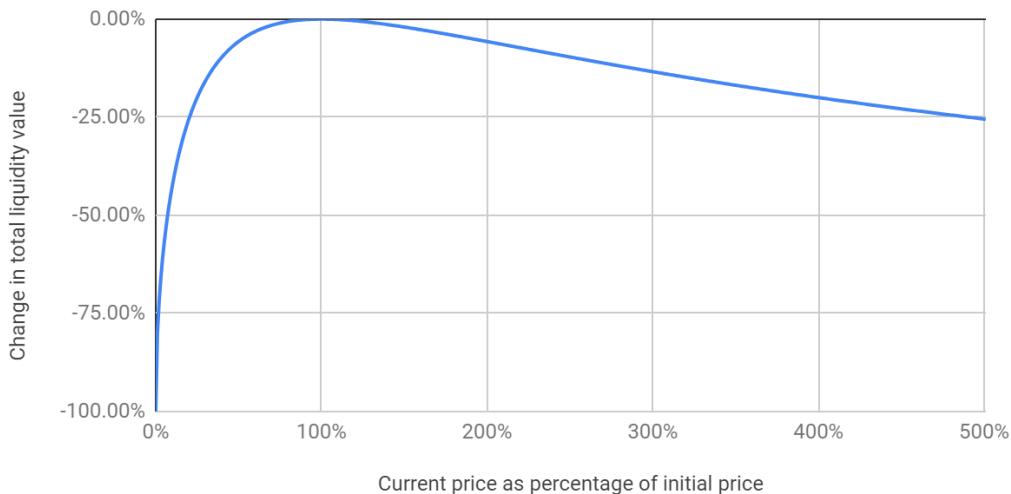
"Of course, if the price were to return to the same value as when the liquidity provider added their liquidity, this loss would disappear. **For this reason, we can call it an impermanent loss.** Using the equations above, we can derive a formula for the size of the impermanent loss in terms of the price ratio between when liquidity was supplied and now. We get the following:"

- `"impermanent_loss = 2 * sqrt(price_ratio) / (1+price_ratio) - 1"`

- "Which we can plot out to get a general sense of the scale of the impermanent loss at different price ratios:"

Losses to liquidity providers due to price variation

Compared to holding the original funds supplied



- "Or to put it another way:"
 - "a 1.25x price change results in a 0.6% loss relative to HODL"
 - "a 1.50x price change results in a 2.0% loss relative to HODL"
 - "a 1.75x price change results in a 3.8% loss relative to HODL"
 - "a 2x price change results in a 5.7% loss relative to HODL"
 - "a 3x price change results in a 13.4% loss relative to HODL"
 - "a 4x price change results in a 20.0% loss relative to HODL"
 - "a 5x price change results in a 25.5% loss relative to HODL"
- "N.B. The loss is the same whichever direction the price change occurs in (i.e. a doubling in price results in the same loss as a halving)." -->

--- id: security title: Security ---

Audit & Formal Verification

Between January 8 and April 30, a team of six engineers reviewed and formally verified crucial components of the smart contracts for Uniswap V2.

Their past work includes smart contract development on and formal verification of multi-collateral DAI.

The scope of work includes:

- Formal verification of the core smart contracts
- Code review of core smart contracts
- Numerical error analysis
- Code review of periphery smart contracts (during ongoing development)

The report also has a "Design Comments" section that we highly recommend for gaining a deep technical understanding of some of the choices made in Uniswap V2.

[Read the report](#)

Bug Bounty

Uniswap has an open and ongoing bug [bounty program](#).

Considerations when building on Uniswap

When integrating Uniswap V2 into another on-chain system, particular care must be taken to avoid security vulnerabilities, avenues for manipulations, and the potential loss of funds.

As a preliminary note: smart contract integrations can happen at two levels: directly with [Pair](#) contracts, or through the [Router](#). Direct interactions offer maximal flexibility but require the most work to get right. Mediated interactions offer more limited capabilities but stronger safety guarantees.

There are two primary categories of risk associated with Uniswap V2. The first involves so-called "static" errors. These can include sending too many tokens to a pair during a swap (or requesting too few tokens back) or allowing transactions to linger in the mempool long enough for the sender's expectations about prices to no longer be accurate.

One may address these errors with fairly straightforward logic checks. Executing these logic checks is the primary purpose of routers. Those who interact directly with pairs must perform these checks themselves (with the help of the [Library](#)).

"Dynamic" risk, the second category, involves runtime pricing. Because Ethereum transactions occur in an adversarial environment, naively written smart contracts can, and will, be exploited for profit. For example, suppose a smart contract checks the asset ratio in a Uniswap pool at runtime and trades against it, assuming that the ratio represents the "fair" or "market" price of these assets. In that case, it is highly vulnerable to manipulation. A malicious actor could, e.g., trivially insert transactions before and after the naive transaction (a so-called "sandwich" attack), causing the smart contract to trade at a radically worse price, profit from this at the trader's expense, and then return the contracts to their original state, all at a low cost. (One important caveat is that these types of attacks are mitigated by trading in highly liquid pools, or at low values.)

The best way to protect against these attacks is to introduce a price oracle. An oracle is any device that returns desired information, in this case, a pair's spot price. The best "oracle" is simply a traders' off-chain observation of the prevailing price, which can be passed into the trade as a safety check. This strategy is best suited to retail trading venues where users initiate transactions on their own behalf. However, it is often the case that a trusted price observation is not available (e.g., in multi-step, programmatic interactions involving Uniswap). Without a price oracle, these interactions will be forced to trade at whatever the (potentially manipulated) rate on Uniswap is. For details on the Uniswap V2 approach to oracles, see [Oracles](#).

id: math title: Math

This section will be expanded in the future. In the mean time, the [Uniswap V2 whitepaper](#) has most relevant math for Uniswap V2.

id: research title: Research

The automated market maker is a new concept, and as such, new research comes out frequently. We've selected some of the most thoughtful here.

Uniswap's Financial Alchemy

Authors: Dave White, Martin Tassy, Charlie Noyes, and Dan Robinson

An automated market maker is a type of decentralized exchange that lets customers trade between on-chain assets like USDC and ETH. Uniswap is the most popular AMM on Ethereum. Like most AMMs, Uniswap facilitates trading between a particular pair of assets by holding reserves of both assets. It sets the trading price between them based on the size of its reserves in such a way that prices will stay in line with the broader market. Anybody who would like to can join the "pool" for a particular pair and become a liquidity provider, or LP, so-called because they provide liquid assets for others to trade against. LPs contribute assets to both reserves simultaneously, taking on some of the risk of trading in exchange for a share of the returns.

- [Uniswap's Financial Alchemy](#)

An analysis of Uniswap markets

Authors: Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, Tarun Chitra

Uniswap---and other constant product markets---appear to work well in practice despite their simplicity. In this paper, we give a simple formal analysis of constant product markets and their generalizations, showing that, under some common conditions, these markets must closely track the reference market price. We also show that Uniswap satisfies many other desirable properties and numerically demonstrate, via a large-scale agent-based simulation, that Uniswap is stable under a wide range of market conditions.

- [An analysis of Uniswap markets](#)

Improved Price Oracles: Constant Function Market Makers

Authors: Guillermo Angeris, Tarun Chitra

Automated market makers, first popularized by Hanson's logarithmic market scoring rule (or LMSR) for prediction markets, have become important building blocks, called 'primitives,' for decentralized finance. A particularly useful primitive is the ability to measure the price of an asset, a problem often known as the pricing oracle problem. In this paper, we focus on the analysis of a very large class of automated market makers, called constant function market makers (or CFMMs) which includes existing popular market makers such as Uniswap, Balancer, and Curve, whose yearly transaction volume totals to billions of dollars. We give sufficient conditions such that, under fairly general assumptions, agents who interact with these constant function market makers are incentivized to correctly report the price of an asset and that they can do so in a computationally efficient way. We also derive several other useful properties that were previously not known. These include lower bounds on the total value of assets held by CFMMs and lower bounds guaranteeing that no agent can, by any set of trades, drain the reserves of assets held by a given CFMM.

- [Improved Price Oracles: Constant Function Market Makers](#)

Pintail research

Published [medium](#) articles by Pintail.

- [Understanding Uniswap Returns](#)
- [Uniswap: A Good Deal for Liquidity Providers?](#)

Liquidity Provider Returns in Geometric Mean Markets

Authors: Alex Evans

Geometric mean market makers (G3Ms), such as Uniswap and Balancer, comprise a popular class of automated market makers (AMMs) defined by the following rule: the reserves of the AMM before and after each trade must have the same (weighted) geometric mean. This paper extends several results known for constant-weight G3Ms to the general case of G3Ms with time-varying and potentially stochastic weights. These results include the returns and no-arbitrage prices of liquidity pool (LP) shares that investors receive for supplying liquidity to G3Ms. Using these expressions, we show how to create G3Ms whose LP shares replicate the payoffs of financial derivatives. The resulting hedges are model-independent and exact for derivative contracts whose payoff functions satisfy an elasticity constraint. These strategies allow LP shares to replicate various trading strategies and financial contracts, including standard options. G3Ms are thus shown to be capable of recreating a variety of active trading strategies through passive positions in LP shares.

- [Liquidity Provider Returns in Geometric Mean Markets](#)

The Replicating Portfolio of a Constant Product Market

Authors: Joseph Clark

We derive the replicating portfolio of a constant product market. This is structurally short volatility (selling options) which explains why positive transaction costs are needed to induce liquidity providers to participate. Where futures and options markets do not exist, this payoff can be used to create them.

- https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3550601

id: using-the-api title: 'Using the API'

In this guide we will create a web interface that consumes and displays data from the Uniswap Subgraph. The goal is to provide a quick overview of a setup that you can extend to create your own UIs and analytics around Uniswap data.

Many different libraries can be used to create an interface and a connection to the subgraph graphql endpoint, but in this guide we will use [React](#) for the interface, and [Apollo Client](#) for sending queries. We'll also be using yarn for dependency management.

Setup and Installs

We'll need to create the basic skeleton for the application. We'll use [create-react-app](#) for this. We'll also add the dependencies we need. Navigate to your root location in your command line and run:

```
yarn create react-app uniswap-demo
cd uniswap-demo
yarn add apollo-client apollo-cache-inmemory apollo-link-http graphql graphql-tag @apollo/react-hooks
yarn start
```

In your browser you should see the default React app running. In a text editor open `App.js` within `src` and replace the contents with this stripped down boilerplate. We'll add to this as we go.

```
import React from 'react'
import './App.css'

function App() {
```

```

    return <div></div>
}

export default App

```

Graphql Client

We need to set up some middleware in order to make requests to the Uniswap subgraph and receive data. To do this we'll use Apollo and create a graphql client to handle this.

1. Add the imports shown below and instantiate a new client instance. Notice how we use the link to the Uniswap subgraph here.

```

import React from 'react'
import './App.css'
import { ApolloClient } from 'apollo-client'
import { InMemoryCache } from 'apollo-cache-inmemory'
import { HttpLink } from 'apollo-link-http'

export const client = new ApolloClient({
  link: new HttpLink({
    uri: 'https://api.thegraph.com/subgraphs/name/uniswap/uniswap-v2',
  }),
  cache: new InMemoryCache(),
})

function App() {
  return <div></div>
}

export default App

```

2. We also need to add a context so that Apollo can handle requests properly. In your `index.js` file import the proper provider and wrap the root in it like this:

```

import React from 'react'
import ReactDOM from 'react-dom'
import App from './App'
import registerServiceWorker from './registerServiceWorker'
import './index.css'
import { ApolloProvider } from 'react-apollo'
import { client } from './App'

ReactDOM.render(
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>,
  document.getElementById('root')
)
registerServiceWorker()

```

Writing the queries

Next we'll construct our query and fetch data. For this example we will fetch some data about the Dai token on Uniswap V2. We'll get the current price, and total liquidity across all pairs. We'll be using the Dai address as an id in this query. We'll also fetch the USD price of ETH to help create USD conversion for Dai data.

1. First we need to define the query itself. We'll use `gql` to parse a query string into the GraphQL AST standard. Import the `gql` helper into the app and use it to create the query. Add the following to your `App.js` file:

```

import gql from 'graphql-tag'

const DAI_QUERY = gql` 
query tokens($tokenAddress: Bytes!) {
  tokens(where: { id: $tokenAddress }) {
    derivedETH
    totalLiquidity
  }
}

```

```

const ETH_PRICE_QUERY = gql`  
query ethPrice {  
  bundle(id: "1") {  
    ethPrice  
  }  
}
```

```

We use an id of `1` for the bundle because there is only one hardcoded bundle in the subgraph.

## Fetch data

Now we're ready to use these queries to fetch data from the Uniswap V2 subgraph. To do this we can use the `useQuery` hook which uses our client instance to fetch data, and gives us live info about the status of the request. To do this add the following to your `App.js` file:

```

import { useQuery } from '@apollo/react-hooks'

const { loading, error, data: ethPriceData } = useQuery(ETH_PRICE_QUERY)
const {
 loading: daiLoading,
 error: daiError,
 data: daiData,
} = useQuery(DAI_QUERY, {
 variables: {
 tokenAddress: '0x6b175474e89094c44da98b954eedeac495271d0f',
 },
})
```

```

Notice we're using the Dai token address to fetch data about Dai.

Formatting Response

Now that we have our data we can format it and display it in the UI. First, we parse the return data to get the actual data that we want. Then we'll use it to get the USD price of Dai. Lastly we'll insert this data into the UI itself.

These queries will return a response object for each query. Within each one we're interested in the root field we defined in the query definition. For the `daiData` response we defined this as `tokens`, and for the `ethPriceData` query we defined this as `ethPrice`. Within each one we'll get an array of results. Because we're only querying for single entities we'll reference the `0` index in the data array.

Add the following lines to your `App.js` file to parse the responses:

```

const daiPriceInEth = daiData && daiData.tokens[0].derivedETH
const daiTotalLiquidity = daiData && daiData.tokens[0].totalLiquidity
const ethPriceInUSD = ethPriceData && ethPriceData.bundles[0].ethPrice
```

```

## Displaying in the UI

Finally we can use our parsed response data to hydrate the UI. We'll do this in two steps.

1. First we'll create loading states. To detect if a query is still pending a response we can reference the loading variables we've already defined. We'll add two loading states, one for the Dai price, and one for the Dai total liquidity. These may flicker fast because the time to query is fast.
2. Populate with loaded data. Once we detect that the queries have finished loading we can populate the UI with the real data.

To do this add the following lines in the return function of your `App.js` file:

```

return (
 <div>
 <div>
 Dai price:{' '}
 {ethLoading || daiLoading
 ? 'Loading token data...'
 : '$' +
 // parse responses as floats and fix to 2 decimals
 (parseFloat(daiPriceInEth) * parseFloat(ethPriceInUSD)).toFixed(2)}
 </div>
 <div>
```

```

```

        Dai total liquidity:' '}
        {daiLoading
          ? 'Loading token data...'
          : // display the total amount of DAI spread across all pools
            parseFloat(daiTotalLiquidity).toFixed(0)}
      </div>
    </div>
)

```

Next steps

This should render a very basic page with these two stats about the Dai token within Uniswap. This is a very basic example of what you can do with the Uniswap subgraph and we encourage you to build out more complex and interesting tools!

You can visit our [analytics site](#) to see a more advanced analytics page, or visit [the github](#) for more detailed examples of using the Uniswap subgraph to create UIs.

Review

In the end your `App.js` file should look like this:

```

import React, { useEffect } from 'react'
import './App.css'
import { ApolloClient } from 'apollo-client'
import { InMemoryCache } from 'apollo-cache-inmemory'
import { HttpLink } from 'apollo-link-http'
import { useQuery } from '@apollo/react-hooks'
import gql from 'graphql-tag'

export const client = new ApolloClient({
  link: new HttpLink({
    uri: 'https://api.thegraph.com/subgraphs/name/uniswap/uniswap-v2',
  }),
  fetchOptions: {
    mode: 'no-cors',
  },
  cache: new InMemoryCache(),
})

const DAI_QUERY = gql` 
query tokens($tokenAddress: Bytes!) {
  tokens(where: { id: $tokenAddress }) {
    derivedETH
    totalLiquidity
  }
}
` 

const ETH_PRICE_QUERY = gql` 
query bundles {
  bundles(where: { id: "1" }) {
    ethPrice
  }
}
` 

function App() {
  const { loading: ethLoading, data: ethPriceData } = useQuery(ETH_PRICE_QUERY)
  const { loading: daiLoading, data: daiData } = useQuery(DAI_QUERY, {
    variables: {
      tokenAddress: '0x6b175474e89094c44da98b954eedeac495271d0f',
    },
  })

  const daiPriceInEth = daiData && daiData.tokens[0].derivedETH
  const daiTotalLiquidity = daiData && daiData.tokens[0].totalLiquidity
  const ethPriceInUSD = ethPriceData && ethPriceData.bundles[0].ethPrice

  return (
    <div>

```

```

<div>
  Dai price:{' '}
  {ethLoading || daiLoading
    ? 'Loading token data...'
    : '$' +
      // parse responses as floats and fix to 2 decimals
      (parseFloat(daiPriceInEth) * parseFloat(ethPriceInUSD)).toFixed(2)}
</div>
<div>
  Dai total liquidity:{' '}
  {daiLoading
    ? 'Loading token data...'
    : // display the total amount of DAI spread across all pools
      parseFloat(daiTotalLiquidity).toFixed(0)}
</div>
</div>
)
}

export default App

```

id: custom-interface-linking title: Custom Linking

Query Parameters

The Uniswap front-end supports URL query parameters to allow for custom linking to the Uniswap frontend. Users and developers can use these query parameters to link to the Uniswap frontend with custom prefilled settings.

Each Page has specific available URL parameters that can be set. Global parameters can be used on all pages.

A parameter used on an incorrect page will have no effect on frontend settings. Parameters not set with a URL parameter will be set to standard frontend defaults.

Global

| Parameter | Type | Description |
|-----------|--------|----------------------------------|
| theme | String | Sets them to dark or light mode. |

Theme Options

Theme can be set as `light` or `dark`.

Example Usage

<https://app.uniswap.org/#/swap?theme=dark>

Swap Page

| Parameter | Type | Description |
|----------------|----------------|---|
| inputCurrency | address | Input currency that will be swapped for output currency. |
| outputCurrency | address or ETH | Output currency that input currency will be swapped for. |
| exactAmount | number | The custom token amount to buy or sell. |
| exactField | string | The field to set custom token amount for. Must be <code>input</code> or <code>output</code> . |

Defaults

ETH defaults as the input currency. When a different token is selected for either input or output ETH will default as the opposite selected currency.

Constraints

Addresses must be valid ERC20 addresses. Slippage and amount values must be valid numbers accepted by the frontend (or error will prevent from swapping). Slippage can 0, or within the range 10->9999 bips (which converts to 0%, 0.01%->99%)

When selecting ETH as the output currency a user must also choose an `inputCurrency` that is not ETH (to prevent ETH being populated in both fields)

Setting Amounts

Two parameters, exactField and exactAmount can be used to set specific token amounts to be sold or bought. Both fields must be set in the URL or there will be no effect on the settings.

Example Usage

```
https://app.uniswap.org/#/swap?  
exactField=input&exactAmount=10&inputCurrency=0x0F5D2fB29fb7d3CFeE444a200298f468908cC942
```

Pool Page

The Pool page is made up of 2 subroutines: `add` , `remove` .

Add Liquidity

| Parameter | Type | Description |
|-----------|---------|--|
| Token0 | address | Pool to withdraw liquidity from. (Must be an ERC20 address with an existing token) |
| Token1 | address | Pool to withdraw liquidity from. (Must be an ERC20 address with an existing token) |

Example Usage

```
https://app.uniswap.org/#/add/v2/0x6B175474E89094C44Da98b954EedeAC495271d0F/0xdAC17F958D2ee523a2206206994597C13D831ec7
```

Remove Liquidity

| Parameter | Type | Description |
|-----------|---------|--|
| Token0 | address | Pool to withdraw liquidity from. (Must be an ERC20 address with an existing token) |
| Token1 | address | Pool to withdraw liquidity from. (Must be an ERC20 address with an existing token) |

Example Usage

```
https://app.uniswap.org/#/remove/0x6B175474E89094C44Da98b954EedeAC495271d0F-  
0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
```

id: quick-start title: Smart Contract Quick start

Developing smart contracts for Ethereum involves a bevy of off-chain tools used for producing and testing bytecode that runs on the [Ethereum Virtual Machine \(EVM\)](#). Some tools also include workflows for deploying this bytecode to the Ethereum network and testnets. There are many options for these tools. This guide walks you through writing and testing a simple smart contract that interacts with the Uniswap Protocol using one specific set of tools (`truffle` + `npm` + `mocha`).

Requirements

To follow this guide, you must have the following installed:

- [nodejs >= v12.x & npm >= 6.x](#)

Bootstrapping a project

You can start from scratch, but it's easier to use a tool like `truffle` to bootstrap an empty project. Create an empty directory and run `npx truffle init` inside that directory to unbox the default [Truffle box](#).

```
mkdir demo  
cd demo  
npx truffle init
```

Setting up npm

In order to reference the Uniswap V2 contracts, you should use the npm artifacts we deploy containing the core and periphery smart contracts and interfaces. To add npm dependencies, we first initialize the npm package. We can run `npm init` in the same directory to create a `package.json` file. You can accept all the defaults and change it later.

```
npm init
```

Adding dependencies

Now that we have an npm package, we can add our dependencies. Let's add both the [@uniswap/v2-core](#) and [@uniswap/v2-periphery](#) packages.

```
npm i --save @uniswap/v2-core
npm i --save @uniswap/v2-periphery
```

If you check the `node_modules/@uniswap` directory, you can now find the Uniswap V2 contracts.

```
moody@MacBook-Pro ~/u/demo> ls node_modules/@uniswap/v2-core/contracts
UniswapV2ERC20.sol      UniswapV2Pair.sol      libraries/
UniswapV2Factory.sol    interfaces/           test/
moody@MacBook-Pro ~/u/demo> ls node_modules/@uniswap/v2-periphery/contracts/
UniswapV2Migrator.sol   examples/            test/
UniswapV2Router01.sol   interfaces/
UniswapV2Router02.sol   libraries/
```

These packages include both the smart contract source code and the build artifacts.

Writing our contract

We can now get started writing our example contract. For writing Solidity, we recommend IntelliJ or VSCode with a solidity plugin, but you can use any text editor. Let's write a contract that returns the value of some amount of liquidity shares for a given token pair. First create a couple of files:

```
mkdir contracts/interfaces
touch contracts/interfaces/ILiquidityValueCalculator.sol
touch contracts/LiquidityValueCalculator.sol
```

This will be the interface of the contract we implement. Put it in `contracts/interfaces/ILiquidityValueCalculator.sol`.

```
pragma solidity ^0.6.6;

interface ILiquidityValueCalculator {
    function computeLiquidityShareValue(uint liquidity, address tokenA, address tokenB) external returns (uint tokenAAmount, uint tokenBAmount);
}
```

Now let's start with the constructor. You need to know where the `UniswapV2Factory` is deployed in order to compute the address of the pair and look up the total supply of liquidity shares, plus the amounts for the reserves. We can store this as an address passed to the constructor.

The factory address is constant on mainnet and all testnets, so it may be tempting to make this value a constant in your contract, but since we need to unit test the contract it should be an argument. You can use solidity immutables to save on gas when accessing this variable.

```
pragma solidity ^0.6.6;

import './interfaces/ILiquidityValueCalculator.sol';

contract LiquidityValueCalculator is ILiquidityValueCalculator {
    address public factory;
    constructor(address factory_) public {
        factory = factory_;
    }
}
```

Now we need to be able to look up the total supply of liquidity for a pair, and its token balances. Let's put this in a separate function. To implement it, we must:

1. Look up the pair address
2. Get the reserves of the pair
3. Get the total supply of the pair liquidity
4. Sort the reserves in the order of tokenA, tokenB

The [UniswapV2Library](#) has some helpful methods for this.

```
pragma solidity ^0.6.6;
```

```

import './interfaces/ILiquidityValueCalculator.sol';
import '@uniswap/v2-periphery/contracts/libraries/UniswapV2Library.sol';
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol';

contract LiquidityValueCalculator is ILiquidityValueCalculator {
    function pairInfo(address tokenA, address tokenB) internal view returns (uint reserveA, uint reserveB, uint totalSupply) {
        IUniswapV2Pair pair = IUniswapV2Pair(UniswapV2Library.pairFor(factory, tokenA, tokenB));
        totalSupply = pair.totalSupply();
        (uint reserves0, uint reserves1,) = pair.getReserves();
        (reserveA, reserveB) = tokenA == pair.token0() ? (reserves0, reserves1) : (reserves1, reserves0);
    }
}

```

Finally we just need to compute the share value. We will leave that as an exercise to the reader.

```

pragma solidity ^0.6.6;

import './interfaces/ILiquidityValueCalculator.sol';
import '@uniswap/v2-periphery/contracts/libraries/UniswapV2Library.sol';
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol';

contract LiquidityValueCalculator is ILiquidityValueCalculator {
    address public factory;
    constructor(address factory_) public {
        factory = factory_;
    }

    function pairInfo(address tokenA, address tokenB) internal view returns (uint reserveA, uint reserveB, uint totalSupply) {
        IUniswapV2Pair pair = IUniswapV2Pair(UniswapV2Library.pairFor(factory, tokenA, tokenB));
        totalSupply = pair.totalSupply();
        (uint reserves0, uint reserves1,) = pair.getReserves();
        (reserveA, reserveB) = tokenA == pair.token0() ? (reserves0, reserves1) : (reserves1, reserves0);
    }

    function computeLiquidityShareValue(uint liquidity, address tokenA, address tokenB) external override returns (uint tokenAAmount, uint tokenBAmount) {
        revert('TODO');
    }
}

```

Writing tests

In order to test your contract, you need to:

1. Bring up a testnet
2. Deploy the `UniswapV2Factory`
3. Deploy at least 2 ERC20 tokens for a pair
4. Create a pair for the factory
5. Deploy your `LiquidityValueCalculator` contract
6. Call `LiquidityValueCalculator#computeLiquidityShareValue`
7. Verify the result with an assertion

#1 is handled for you automatically by the `truffle test` command.

Note you should only deploy the precompiled Uniswap contracts in the `build` directories for unit tests. This is because solidity appends a metadata hash to compiled contract artifacts which includes the hash of the contract source code path, and compilations on other machines will not result in the exact same bytecode. This is problematic because in Uniswap V2 we use the hash of the bytecode in the v2-periphery `UniswapV2Library`, to compute the pair address.

To get the bytecode for deploying `UniswapV2Factory`, you can import the file via:

```
const UniswapV2FactoryBytecode = require('@uniswap/v2-core/build/UniswapV2Factory.json').bytecode
```

We recommend using a standard ERC20 from `@openzeppelin/contracts` for deploying an ERC20.

You can read more about deploying contracts and writing tests using Truffle [here](#).

Compiling and deploying the contract

Learn more about compiling and deploying contracts using Truffle [here](#) and [here](#) respectively.

WIP

This guide is a WIP. Please contribute to this guide with the edit button below!

id: trading-from-a-smart-contract title: Implement a Swap

When trading from a smart contract, the most important thing to keep in mind is that access to an external price source is *required*. Without this, trades can be frontrun for considerable loss.

Read [safety considerations](#) for more.

Using the Router

The easiest way to safely swap tokens is to use the [router](#), which provides a variety of methods to safely swap to and from different assets. You'll notice that there is a function for each permutation of swapping to/from an exact amount of ETH/tokens.

First you must use an external price source to calculate the safety parameters for the function you'd like to call. This is either a minimum amount received when selling an exact input or the maximum amount you are willing to pay when buying an exact output amount.

It is also important to ensure that your contract controls enough ETH/tokens to make the swap, and has granted approval to the router to withdraw this many tokens.

Check out the [Pricing](#) page for a more in depth discussion on getting prices.

Example

Imagine you want to swap 50 DAI for as much ETH as possible from your smart contract.

transferFrom

Before swapping, our smart contracts needs to be in control of 50 DAI. The easiest way to accomplish this is by calling `transferFrom` on DAI with the owner set to `msg.sender`:

```
uint amountIn = 50 * 10 ** DAI.decimals();
require(DAI.transferFrom(msg.sender, address(this), amountIn), 'transferFrom failed.');
```

approve

Now that our contract owns 50 DAI, we need to approve to the [router](#) to withdraw this DAI:

```
require(DAI.approve(address(UniswapV2Router02), amountIn), 'approve failed.');
```

swapExactTokensForETH

Now we're ready to swap:

```
// amountOutMin must be retrieved from an oracle of some kind
address[] memory path = new address[](2);
path[0] = address(DAI);
path[1] = UniswapV2Router02.WETH();
UniswapV2Router02.swapExactTokensForETH(amountIn, amountOutMin, path, msg.sender, block.timestamp);
```

Safety Considerations

Because Ethereum transactions occur in an adversarial environment, smart contracts that do not perform safety checks *can be exploited for profit*. If a smart contract assumes that the current price on Uniswap is a "fair" price without performing safety checks, *it is vulnerable to manipulation*. A bad actor could e.g. easily insert transactions before and after the swap (a "sandwich" attack) causing the smart contract to trade at a much worse price, profit from this at the trader's expense, and then return the contracts to their original state. (One important caveat is that these types of attacks are mitigated by trading in extremely liquid pools, and/or at low values.)

The best way to protect against these attacks is to use an external price feed or "price oracle". The best "oracle" is simply *traders' off-chain observation of the current price*, which can be passed into the trade as a safety check. This strategy is best for situations where users initiate trades on their own behalf.

However, when an off-chain price can't be used, an on-chain oracle should be used instead. Determining the best oracle for a given situation is a not part of this guide, but for more details on the Uniswap V2 approach to oracles, see [Oracles](#).

id: providing-liquidity title: Providing Liquidity

Providing Liquidity

Introduction

When providing liquidity from a smart contract, the most important thing to keep in mind is that tokens deposited into a pool at any rate other than the current reserve ratio are *vulnerable to being arbitrated*. As an example, if the ratio of x:y in a pair is 10:2 (i.e. the price is 5), and someone naively adds liquidity at 5:2 (a price of 2.5), the contract will simply accept all tokens (changing the price to 3.75 and opening up the market to arbitrage), but only issue pool tokens entitling the sender to the amount of assets sent at the proper ratio, in this case 5:1. To avoid donating to arbitrageurs, it is imperative to add liquidity at the current price. Luckily, it's easy to ensure that this condition is met!

Using the Router

The easiest way to safely add liquidity to a pool is to use the [router](#), which provides simple methods to safely add liquidity to a pool. If the liquidity is to be added to an ERC-20/ERC-20 pair, use [addLiquidity](#). If WETH is involved, use [addLiquidityETH](#).

These methods both require the caller to commit to a *belief about the current price*, which is encoded in the `amount*Desired` parameters. Typically, it's fairly safe to assume that the current fair market price is around what the current reserve ratio is for a pair (because of arbitrage). So, if a user wants to add 1 ETH to a pool, and the current DAI/WETH ratio of the pool is 200/1, it's reasonable to calculate that 200 DAI must be sent along with the ETH, which is an implicit commitment to the price of 200 DAI/1 WETH. However, it's important to note that this must be calculated *before the transaction is submitted*. It is *not safe* to look up the reserve ratio from within a transaction and rely on it as a price belief, as this ratio can be cheaply manipulated to your detriment.

However, it is still possible to submit a transaction which encodes a belief about the price which ends up being wrong because of a larger change in the true market price before the transaction is confirmed. For that reason, it's necessary to pass an additional set of parameters which encode the caller's tolerance to price changes. These `amount*Min` parameters should typically be set to percentages of the calculated desired price. So, at a 1% tolerance level, if our user sends a transaction with 1 ETH and 200 DAI, `amountETHMin` should be set to e.g. .99 ETH, and `amountTokenMin` should be set to 198 DAI. This means that, at worst, liquidity will be added at a rate between 198 DAI/1 ETH and 202.02 DAI/1 ETH (200 DAI/.99 ETH).

Once the price calculations have been made, it's important to ensure that your contract a) controls at least as many tokens/ETH as were passed as `amount*Desired` parameters, and b) has granted approval to the router to withdraw this many tokens.

id: building-an-oracle title: Building an Oracle

To build a price oracle on Uniswap V2, you must first understand the requirements for your use case. Once you understand the kind of price average you require, it is a matter of storing the cumulative price variable from the pair as often as necessary, and computing the average price using two or more observations of the cumulative price variables.

Understanding requirements

To understand your requirements, you should first research the answer to the following questions:

- Is data freshness important? I.e.: must the price average include the current price?
- Are recent prices more important than historical prices? I.e.: is the current price given more weight than historical prices?

Note your answers for the following discussion.

Oracle Strategies

Fixed windows

In the case where data freshness is not important and recent prices are weighted equally with historical prices, it is enough to store the cumulative price once per period (e.g. once per 24 hours.)

Computing the average price over these data points gives you 'fixed windows', which can be updated after the lapse of each period. We wrote an example oracle of this kind [here](#).

This example does not limit the maximum size of the fixed window, i.e. it only requires that the window size is greater than 1 period (e.g. 24 hours).

Moving averages

In the case where data freshness is important, you can use a sliding window in which the cumulative price variable is measured more often than once per period.

There are at least [two kinds of moving averages](#) that you can compute using the Uniswap cumulative price variable.

[Simple moving averages](#) give equal weight to each price measurement. We have built an example of a sliding window oracle [here](#).

[Exponential moving averages](#) give more weight to the most recent price measurements. We do not yet have an example written for this type of oracle.

You may wish to use exponential moving averages where recent prices are more important than historical prices, e.g. in case of liquidations. However, note that putting more weight on recent prices makes the oracle cheaper to manipulate than weighting all price measurements equally.

Computing average prices

To compute the average price given two cumulative price observations, take the difference between the cumulative price at the beginning and end of the period, and divide by the elapsed time between them in seconds. This will produce a [fixed point unsigned Q112x112](#) number that represents the price of one asset relative to the other. This number is represented as a `uint224` where the upper 112 bits represent the integer amount, and the lower 112 bits represent the fractional amount.

Pairs contain both `price0CumulativeLast` and `price1CumulativeLast`, which are ratios of reserves of `token1 / token0` and `token0 / token1` respectively. I.e. the price of `token0` is expressed in terms of `token1 / token0`, while the price of `token1` is expressed in terms of `token0 / token1`.

Getting the latest cumulative price

If you wish to compute the average price between a historical price cumulative observation and the current cumulative price, you should use the cumulative price values from the current block. If the cumulative price has not been updated in the current block, e.g. because there has not been any liquidity event (`mint` / `burn` / `swap`) on the pair in the current block, you can compute the cumulative price counterfactually.

We provide a library for use in oracle contracts that has the method [UniswapV2OracleLibrary#currentCumulativePrices](#) for getting the cumulative price as of the current block. The current cumulative price returned by this method is computed *counterfactually*, meaning it requires no call to the relative gas-expensive `#sync` method on the pair. It is correct regardless of whether a swap has already executed in the current block.

Notes on overflow

The `UniswapV2Pair` cumulative price variables are designed to eventually overflow, i.e. `price0CumulativeLast` and `price1CumulativeLast` and `blockTimestampLast` will overflow through 0.

This should not pose an issue to your oracle design, as the price average computation is concerned with differences (i.e. subtraction) between two separate observations of a cumulative price variable. Subtracting between two cumulative price values will result in a number that fits within the range of `uint256` as long as the observations are made for periods of max `2^32` seconds, or ~136 years.

`blockTimestampLast` is stored only in a `uint32`. For the same reason as described above, the pair can save a storage slot, and many SSTORES over the life of the pair, by storing only `block.timestamp % uint32(-1)`. This is feasible because the pair is only concerned with the time that elapses between each liquidity event when updating the cumulative prices, which is always expected to be less than `2^32` seconds.

When computing time elapsed within your own oracle, you can simply store the `block.timestamp` of your observations as `uint256`, and avoid dealing with overflow math for computing the time elapsed between observations. This is how the [ExampleSlidingWindowOracle](#) handles observation timestamps.

Integrating the oracle

To integrate an oracle into your contracts, you must ensure the oracle's observations of the cumulative price variable are kept up to date. As long as your oracle is up to date, you can depend on it to produce average prices. The process of keeping your oracle up to date is called 'maintenance'.

Oracle maintenance

In order to measure average prices over a period, the oracle must have a way of referencing the cumulative price at the start and end of a period. The recommended way of doing this is by storing these prices in the oracle contract, and calling the oracle frequently enough to store the latest cumulative price.

Reliable oracle maintenance is a difficult task, and can become a point of failure in times of congestion. Instead, consider building this functionality directly into the critical calls of your own smart contracts, or incentivize oracle maintenance calls by other parties.

No-maintenance option

It is possible to avoid regularly storing this cumulative price at the start of the period by utilizing storage proofs. However, this approach has limitations, especially in regard to gas cost and maximum length of the time period over which the average price can be measured. If you wish to try this approach, you can follow [this repository by Keydonix](#).

Keydonix has developed a general purpose price feed oracle built on Uniswap v2 that supports arbitrary time windows (up to 256 blocks) and doesn't require any active maintenance.

id: using-flash-swaps title: Flash Swaps

Flash swaps are an integral feature of Uniswap V2. In fact, under the hood, all swaps are actually flash swaps! This simply means that pair contracts send output tokens to the recipient *before* enforcing that enough input tokens have been received. This is slightly atypical, as one might expect a pair to ensure it's received payment before delivery. However, because Ethereum transactions are *atomic*, we can roll back the entire swap if it turns out that the contract hasn't received enough tokens to make itself whole by the end of the transaction.

To see how this all works, let's start by examining the interface of the `swap` function:

```
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data);
```

For the sake of example, let's assume that we're dealing with a DAI/WETH pair, where DAI is `token0` and WETH is `token1`. `amount0Out` and `amount1Out` specify the amount of DAI and WETH that the `msg.sender` wants the pair to send to the `to` address (one of these amounts may be 0). At this point you may be wondering how the contract receives tokens. For a typical (non-flash) swap, it's actually the responsibility of `msg.sender` to ensure that enough WETH or DAI has *already been sent* to the pair before `swap` is called (in the context of trading, this is all handled neatly by a router contract). But when executing a flash swap, *tokens do not need to be sent to the contract before calling swap*. Instead, they must be sent from within a *callback function* that the pair triggers on the `to` address.

Triggering a Flash Swap

To differentiate between the "typical" trading case and the flash swap case, pairs use the `data` parameter. Specifically, if `data.length` equals 0, the contract assumes that payment has already been received, and simply transfers the tokens to the `to` address. But, if `data.length` is greater than 0, the contract transfers the tokens and then calls the following function on the `to` address:

```
function uniswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data);
```

The logic behind this identification strategy is simple: the vast majority of valid flash swap use cases involve interactions with external protocols. The best way to pass information dictating how these interactions happen (function arguments, safety parameters, addresses, etc.) is via the `data` parameter. It's expected that `data` will be `abi.decode`d from within `uniswapV2Call`. In the rare case where no data is required, callers should ensure that `data.length` equals 1 (i.e. encode a single junk byte as `bytes`), and then ignore this argument in `uniswapV2Call`.

Pairs call `uniswapV2Call` with the `sender` argument set to the `msg.sender` of the `swap`. `amount0` and `amount1` are simply `amount0Out` and `amount1Out`.

Using uniswapV2Call

There are several conditions that should be checked in all `uniswapV2Call` functions:

```
function uniswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data) {
    address token0 = IUniswapV2Pair(msg.sender).token0(); // fetch the address of token0
    address token1 = IUniswapV2Pair(msg.sender).token1(); // fetch the address of token1
    assert(msg.sender == IUniswapV2Factory(factoryV2).getPair(token0, token1)); // ensure that msg.sender is a V2
pair
    // rest of the function goes here!
}
```

The first 2 lines simply fetch the token addresses from the pair, and the 3rd ensures that the `msg.sender` is an actual Uniswap V2 pair address.

Repayment

At the end of `uniswapV2Call`, contracts must return enough tokens to the pair to make it whole. Specifically, this means that the product of the pair reserves after the swap, discounting all token amounts sent by 0.3% LP fee, must be greater than before.

Multi-Token

In the case where the token withdrawn is *not* the token returned (i.e. DAI was requested in the flash swap, and WETH was returned, or vice versa), the fee simplifies to the simple swap case. This means that the standard `getAmountIn` pricing function should be used to calculate e.g., the amount of WETH that must be returned in exchange for the amount of DAI that was requested out.

This type of fee calculation gives a slight advantage to the caller, as the fee derived from repayment in a corresponding token will always be slightly less than the fee derived from a direct token repayment, as a result of the difference between the amount required to pay back a swap, versus the amount withdrawn and then directly returned. The approximate comparison of fees is ~ 30 bps for a swap fee vs. 30.09 bps for a direct repayment.

Single-Token

In the case where the token withdrawn is the *same* as the token returned (i.e. DAI was requested in the flash swap, used, then returned, or vice versa with WETH), the following condition must be satisfied:

```
DAIReservePre - DAIWithdrawn + (DAIReturned * .997) >= DAIReservePre
```

It may be more intuitive to rewrite this formula in terms of a "fee" levied on the *withdrawn* amount (despite the fact that Uniswap always levies fees on input amounts, in this case the *returned* amount, here we can simplify to an effective fee on the *withdrawn* amount). If we rearrange, the formula looks like:

```
(DAIReturned * .997) - DAIWithdrawn >= 0
```

```
DAIReturned >= DAIWithdrawn / .997
```

So, the effective fee on the withdrawn amount is $.003 / .997 \approx 0.3009027\%$.

Resources

For further exploration of flash swaps, see the [whitepaper](#).

Example

A fully functional example of flash swaps is available: [ExampleFlashSwap.sol](#).

Interface

```
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Callee.sol';

pragma solidity >=0.5.0;

interface IUniswapV2Callee {
    function uniswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data) external;
}
```

id: getting-pair-addresses title: Pair Addresses

getPair

The most obvious way to get the address for a pair is to call `getPair` on the factory. If the pair exists, this function will return its address, else `address(0)` (`0x000`).

- The "canonical" way to determine whether or not a pair exists.
- Requires an on-chain lookup.

CREATE2

Thanks to some [fancy footwork in the factory](#), we can also compute pair addresses *without any on-chain lookups* because of [CREATE2](#). The following values are required for this technique:

| | |
|---------|-------------------------------------|
| address | The factory address |
|---------|-------------------------------------|

| | |
|----------------------|---|
| salt | <code>keccak256(abi.encodePacked(token0, token1))</code> |
| keccak256(init_code) | <code>0x96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f</code> |

- `token0` must be strictly less than `token1` by sort order.
- Can be computed offline.
- Requires the ability to perform `keccak256`.

Examples

Solidity

```
address factory = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;
address token0 = 0xCAFE00000000000000000000000000000000000000000000000000000; // change me!
address token1 = 0xF00D0000000000000000000000000000000000000000000000000000; // change me!

address pair = address(uint(keccak256(abi.encodePacked(
    hex'ff',
    factory,
    keccak256(abi.encodePacked(token0, token1)),
    hex'96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f'
))));
```

id: supporting-meta-transactions title: Supporting meta transactions

All Uniswap V2 pool tokens support meta-transaction approvals via the [permit](#) function. This obviates the need for a blocking approve transaction before programmatic interactions with pool tokens can occur.

ERC-712

In vanilla ERC-20 token contracts, owners may only register approvals by directly calling a function which uses `msg.sender` to permission itself. With meta-approvals, ownership and permissioning are derived from a signature passed into the function by the caller (sometimes referred to as the relayer). Because signing data with Ethereum private keys can be a tricky endeavor, Uniswap V2 relies on [ERC-712](#), a signature standard with widespread community support, to ensure user safety and wallet compatibility.

Domain Separator

```
keccak256(
    abi.encode(
        keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)'),
        keccak256(bytes(name)),
        keccak256(bytes('1')),
        chainId,
        address(this)
    )
);
```

- `name` is always Uniswap V2, see [name](#).
- `chainId` is determined from the [ERC-1344](#) `chainid` opcode.
- `address(this)` is the address of the pair, see [Pair Addresses](#).

Permit Typehash

```
keccak256('Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)');
```

id: overview title: Overview sidebar_position: 1

The Uniswap V2 Smart Contracts

Welcome to the Uniswap protocol V2 docs.

The pages here contain conceptual and technical documentation of the Uniswap V2 protocol.

If you are new to Uniswap, you might want to check out the [Protocol overview](#) first.

You can also take a look at the V2 Protocol [Whitepaper](#).

Developer links

The V2 Uniswap protocol is separated across two repositories

- [uniswap-v2-core](#)
- [uniswap-v2-periphery](#)

The V2 SDK, which can assist developers when interacting with the Uniswap V2 Protocol can be found here.

- [uniswap-sdk](#)
- [uniswap-sdk-core](#)

id: overview title: API Overview

This section explains the Uniswap Subgraph and how to interact with it. The Uniswap subgraph indexes data from the Uniswap contracts over time. It organizes data about pairs, tokens, Uniswap as a whole, and more. The subgraph updates any time a transaction is made on Uniswap. The subgraph runs on [The Graph](#) protocol's hosted service and can be openly queried.

Resources

[Subgraph Explorer](#) - sandbox for querying data and endpoints for developers.

[Uniswap V2 Subgraph](#) - source code for deployed subgraph.

Usage

The subgraph provides a snapshot of the current state of Uniswap and also tracks historical data. It is currently used to power [uniswap.info](#). It is not intended to be used as a data source for structuring transactions (contracts should be referenced directly for the most reliable live data).

Making Queries

To learn more about querying a subgraph refer to [The Graph's documentation](#).

Versions

The [Uniswap V2 Subgraph](#) only tracks data on Uniswap V2. For Uniswap V1 information see the [V1 Subgraph](#).

id: entities title: Entities

Entities define the schema of the subgraph, and represent the data that can be queried. Within each entity are sets of fields that store useful information related to the entity. Below is a list of the available entities within the Uniswap Subgraph, and descriptions for the available fields.

To see an interactive sandbox of all entities see the [Graph Explorer](#).

Each entity is defined with a value type, which will always be a base AssemblyScript type, or a custom type provided by The Graph's custom TypeScript library. For more information on value types see [here](#).

Uniswap Factory

The Uniswap Factory entity is responsible for storing aggregate information across all Uniswap pairs. It can be used to view stats about total liquidity, volume, amount of pairs and more. There is only one UniswapFactory entity in the subgraph.

| Field Name | Value Type | Description |
|-------------------|------------|---|
| id | ID | factory address |
| pairCount | Int | amount of pairs created by the Uniswap factory |
| totalVolumeUSD | BigDecimal | all time USD volume across all pairs (USD is derived) |
| totalVolumeETH | BigDecimal | all time volume in ETH across all pairs (ETH is derived) |
| totalLiquidityUSD | BigDecimal | total liquidity across all pairs stored as a derived USD amount |
| totalLiquidityETH | BigDecimal | total liquidity across all pairs stored as a derived ETH amount |

| | | |
|---------|--------|--|
| txCount | BigInt | all time amount of transactions across all pairs |
|---------|--------|--|

Token

Stores aggregated information for a specific token across all pairs that token is included in.

| Field Name | Value Type | Description |
|--------------------|------------|--|
| id | ID | token address |
| symbol | String | token symbol |
| name | String | token name |
| decimals | BigInt | token decimals |
| tradeVolume | BigDecimal | amount of token traded all time across all pairs |
| tradeVolumeUSD | BigDecimal | amount of token in USD traded all time across pairs (only for tokens with liquidity above minimum threshold) |
| untrackedVolumeUSD | BigDecimal | amount of token in USD traded all time across pairs (no minimum liquidity threshold) |
| txCount | BigInt | amount of transactions all time in pairs including token |
| totalLiquidity | BigDecimal | total amount of token provided as liquidity across all pairs |
| derivedETH | BigDecimal | ETH per token |

Pair

Information about a pair. Includes references to each token within the pair, volume information, liquidity information, and more. The pair entity mirrors the pair smart contract, and also contains aggregated information about use.

| Field Name | Value Type | Description |
|----------------------|---------------------|---|
| id | ID | pair contract address |
| factory | UniswapFactory | reference to Uniswap factory entity |
| token0 | Token | reference to token0 as stored in pair contract |
| token1 | Token | reference to token1 as stored in pair contract |
| reserve0 | BigDecimal | reserve of token0 |
| reserve1 | BigDecimal | reserve of token1 |
| totalSupply | BigDecimal | total supply of liquidity token distributed to LPs |
| reserveETH | BigDecimal | total liquidity in pair stored as an amount of ETH |
| reserveUSD | BigDecimal | total liquidity amount in pair stored as an amount of USD |
| trackedReserveETH | BigDecimal | total liquidity with only tracked amount (see tracked amounts) |
| token0Price | BigDecimal | token0 per token1 |
| token1Price | BigDecimal | token1 per token0 |
| volumeToken0 | BigDecimal | amount of token0 swapped on this pair |
| volumeToken1 | BigDecimal | amount of token1 swapped on this pair |
| volumeUSD | BigDecimal | total amount swapped all time in this pair stored in USD (only tracked if USD liquidity is above minimum threshold) |
| untrackedVolumeUSD | BigDecimal | total amount swapped all time in this pair stored in USD, no minimum liquidity threshold |
| txCount | BigInt | all time amount of transactions on this pair |
| createdAtTimestamp | BigInt | timestamp contract was created |
| createdAtBlockNumber | BigInt | Ethereum block contract was created |
| liquidityPositions | [LiquidityPosition] | array of liquidity providers, used as a reference to LP entities |

User

A user entity is created for any address that provides liquidity to a pool on Uniswap. This entity can be used to track open positions for users. LiquidityPosition entities can be referenced to get specific data about each position.

| Field Name | Value Type | Description |
|--------------------|---------------------|--|
| id | ID | user address |
| liquidityPositions | [LiquidityPosition] | array of all liquidity positions user has open |
| usdSwapped | BigDecimal | total USD value swapped |

LiquidityPosition

This entity is used to store data about a user's liquidity position. This information, along with information from the pair itself can be used to provide position sizes, token deposits, and more.

| Field Name | Value Type | Description |
|-----------------------|------------|--|
| id | ID | user address and pair address concatenated with a dash |
| user | User | reference to user |
| pair | Pair | reference to the pair liquidity is being provided on |
| liquidityTokenBalance | BigDecimal | amount of LP tokens minted for this position |

Transaction

Transaction entities are created for each Ethereum transaction that contains an interaction within Uniswap contracts. This subgraph tracks Mint, Burn, and Swap events on the Uniswap core contracts. Each transaction contains 3 arrays, and at least one of these arrays has a length of 1.

| Field Name | Value Type | Description |
|-------------|------------|---|
| id | ID | Ethereum transaction hash |
| blockNumber | BigInt | block transaction was mined in |
| timestamp | BigInt | timestamp for transaction |
| mints | [Mint] | array of Mint events within the transaction, 0 or greater |
| burns | [Burn] | array of Burn events within transaction, 0 or greater |
| swaps | [Swap] | array of Swap events within transaction, 0 or greater |

Mint

Mint entities are created for every emitted Mint event on the Uniswap core contracts. The Mint entity stores key data about the event like token amounts, who sent the transaction, who received the liquidity, and more. This entity can be used to track liquidity provisions on pairs.

| Field Name | Value Type | Description |
|-------------|-------------|---|
| id | ID | Transaction hash plus index in the transaction mint array |
| transaction | Transaction | reference to the transaction Mint was included in |
| timestamp | BigInt | timestamp of Mint, used to sort recent liquidity provisions |
| pair | Pair | reference to pair |
| to | Bytes | recipient of liquidity tokens |
| liquidity | BigDecimal | amount of liquidity tokens minted |
| sender | Bytes | address that initiated the liquidity provision |
| amount0 | BigDecimal | amount of token0 provided |
| amount1 | BigDecimal | amount of token1 provided |
| logIndex | BigInt | index in the transaction event was emitted |
| amountUSD | BigDecimal | derived USD value of token0 amount plus token1 amount |
| feeTo | Bytes | address of fee recipient (if fee is on) |

| | | |
|--------------|------------|--|
| feeLiquidity | BigDecimal | amount of liquidity sent to fee recipient (if fee is on) |
|--------------|------------|--|

Burn

Burn entities are created for every emitted Burn event on the Uniswap core contracts. The Burn entity stores key data about the event like token amounts, who burned LP tokens, who received tokens, and more. This entity can be used to track liquidity removals on pairs.

| Field Name | Value Type | Description |
|--------------|-------------|---|
| id | ID | Transaction hash plus index in the transaction burn array |
| transaction | Transaction | reference to the transaction Burn was included in |
| timestamp | BigInt | timestamp of Burn, used to sort recent liquidity removals |
| pair | Pair | reference to pair |
| to | Bytes | recipient of tokens |
| liquidity | BigDecimal | amount of liquidity tokens burned |
| sender | Bytes | address that initiated the liquidity removal |
| amount0 | BigDecimal | amount of token0 removed |
| amount1 | BigDecimal | amount of token1 removed |
| logIndex | BigInt | index in the transaction event was emitted |
| amountUSD | BigDecimal | derived USD value of token0 amount plus token1 amount |
| feeTo | Bytes | address of fee recipient (if fee is on) |
| feeLiquidity | BigDecimal | amount of tokens sent to fee recipient (if fee is on) |

Swap

Swap entities are created for each token swap within a pair. The Swap entity can be used to get things like swap size (in tokens and USD), sender, recipient and more. See the Swap overview page for more information on amounts.

| Field Name | Value Type | Description |
|-------------|-------------|---|
| id | ID | transaction hash plus index in Transaction swap array |
| transaction | Transaction | reference to transaction swap was included in |
| timestamp | BigInt | timestamp of swap, used for sorted lookups |
| pair | Pair | reference to pair |
| sender | Bytes | address that initiated the swap |
| amount0In | BigDecimal | amount of token0 sold |
| amount1In | BigDecimal | amount of token1 sold |
| amount0Out | BigDecimal | amount of token0 received |
| amount1Out | BigDecimal | amount of token1 received |
| to | Bytes | recipient of output tokens |
| logIndex | BigInt | event index within transaction |
| amountUSD | BigDecimal | derived amount of tokens sold in USD |

Bundle

The Bundle is used as a global store of derived ETH price in USD. Because there is no guaranteed common base token across pairs, a global reference of USD price is useful for deriving other USD values. The Bundle entity stores an updated weighted average of ETH<->Stablecoin pair prices. This provides a strong estimate for the USD price of ETH that can be used in other places in the subgraph.

| Field Name | Value Type | Description |
|------------|------------|---|
| id | ID | constant 1 |
| ethPrice | BigDecimal | derived price of ETH in USD based on stablecoin pairs |

Historical Entities

The subgraph tracks aggregated information grouped by days to provide insights to daily activity on Uniswap. While [time travel queries](#) can be used for direct comparison against values in the past, it is much more expensive to query grouped data. For this reason the subgraph tracks information grouped in daily buckets, using timestamps provided by contract events. These entities can be used to query things like total volume on a given day, price of a token on a given day, etc.

For each DayData type, a new entity is created each day.

UniswapDayData

Tracks data across all pairs aggregated into a daily bucket.

| Field Name | Value Type | Description |
|-------------------|-----------------|--|
| id | ID | unix timestamp for start of day / 86400 giving a unique day index |
| date | Int | unix timestamp for start of day |
| dailyVolumeETH | BigDecimal | total volume across all pairs on this day, stored as a derived amount of ETH |
| dailyVolumeUSD | BigDecimal | total volume across all pairs on this day, stored as a derived amount of USD |
| totalVolumeETH | BigDecimal | all time volume across all pairs in ETH up to and including this day |
| totalLiquidityETH | BigDecimal | total liquidity across all pairs in ETH up to and including this day |
| totalVolumeUSD | BigDecimal | all time volume across all pairs in USD up to and including this day |
| totalLiquidityUSD | BigDecimal | total liquidity across all pairs in USD up to and including this day |
| maxStored | Int | reference used to store most liquid tokens, used for historical liquidity charts |
| mostLiquidTokens | [TokenDayData!] | tokens with most liquidity in Uniswap |
| txCount | BigInt | number of transactions throughout this day |

Pair Day Data

Tracks pair data across each day.

| Field Name | Value Type | Description |
|-------------------|------------|---|
| id | ID | pair contract address and day id (day start timestamp in unix / 86400) concatenated with a dash |
| date | Int | unix timestamp for start of day |
| pairAddress | Bytes | address for pair contract |
| token0 | Token | reference to token0 |
| token1 | Token | reference to token1 |
| reserve0 | BigDecimal | reserve of token0 (updated during each transaction on pair) |
| reserve1 | BigDecimal | reserve of token1 (updated during each transaction on pair) |
| totalSupply | BigDecimal | total supply of liquidity token distributed to LPs |
| reserveUSD | BigDecimal | reserve of token0 plus token1 stored as a derived USD amount |
| dailyVolumeToken0 | BigDecimal | total amount of token0 swapped throughout day |
| dailyVolumeToken1 | BigDecimal | total amount of token1 swapped throughout day |
| dailyVolumeUSD | BigDecimal | total volume within pair throughout day |
| dailyTxns | BigInt | amount of transactions on pair throughout day |

TokenDayData

Tracks token data aggregated across all pairs that include token.

| Field Name | Value Type | Description |
|------------|------------|---|
| id | ID | token address and day id (day start timestamp in unix / 86400) concatenated with a dash |

| | | |
|---------------------|---------------|--|
| date | Int | unix timestamp for start of day |
| token | Token | reference to token entity |
| dailyVolumeToken | BigDecimal | amount of token swapped across all pairs throughout day |
| dailyVolumeETH | BigDecimal | amount of token swapped across all pairs throughout day stored as a derived amount of ETH |
| dailyVolumeUSD | BigDecimal | amount of token swapped across all pairs throughout day stored as a derived amount of USD |
| dailyTxns | BigInt | amount of transactions with this token across all pairs |
| totalLiquidityToken | BigDecimal | token amount of token deposited across all pairs |
| totalLiquidityETH | BigDecimal | token amount of token deposited across all pairs stored as amount of ETH |
| totalLiquidityUSD | BigDecimal | token amount of token deposited across all pairs stored as amount of USD |
| priceUSD | BigDecimal | price of token in derived USD |
| maxStored | Int | amount of token deposited in pair with highest token liquidity - used only as a reference for storing most liquid pairs for this token |
| mostLiquidPairs | [PairDayData] | pairs with most liquidity for this token |

id: queries title: Queries

The subgraph can be queried to retrieve important information about Uniswap, pairs, tokens, transactions, users, and more. This page will provide examples for common queries.

To try these queries and run your own visit the [subgraph sandbox](#).

Global Data

To query global data you can pass in the Uniswap Factory address and select from available fields.

Global Stats

All time volume in USD, total liquidity in USD, all time transaction count.

```
{
  uniswapFactory(id: "0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f") {
    totalVolumeUSD
    totalLiquidityUSD
    txCount
  }
}
```

Global Historical lookup

To get a snapshot of past state, use The Graph's block query feature and query at a previous block. See this post to get more information about [fetching block numbers from timestamps](#). This can be used to calculate things like 24hr volume.

```
{
  uniswapFactory(id: "0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f", block: {number: 10291203}) {
    totalVolumeUSD
    totalLiquidityUSD
    txCount
  }
}
```

Pair Data

Pair Overview

Fetch a snapshot of the current state of the pair with common values. This example fetches the DAI/WETH pair.

```
{
  pair(id: "0xa478c2975ablea89e8196811f51a7b7ade33eb11") {
    token0 {
      id
      symbol
      name
      derivedETH
    }
  }
}
```

```

    }
    token1 {
      id
      symbol
      name
      derivedETH
    }
    reserve0
    reserve1
    reserveUSD
    trackedReserveETH
    token0Price
    token1Price
    volumeUSD
    txCount
  }
}

```

All pairs in Uniswap

The Graph limits entity return amounts to 1000 per query as of now. To get all pairs on Uniswap use a loop and graphql skip query to fetch multiple chunks of 1000 pairs. The query would look like this (where skip is some incrementing variable passed into your query).

```
{
  query pairs($skip: Int!) {
    pairs(first: 1000, skip: $skip) {
      id
    }
  }
}
```

Most liquid pairs

Order by liquidity to get the most liquid pairs in Uniswap.

```
{
  pairs(first: 1000, orderBy: reserveUSD, orderDirection: desc) {
    id
  }
}
```

Recent Swaps within a Pair

Get the last 100 swaps on a pair by fetching Swap events and passing in the pair address. You'll often want token information as well.

```
{
  swaps(orderBy: timestamp, orderDirection: desc, where: {
    pair: "0xa478c2975ablea89e8196811f51a7b7ade33eb11"
  }) {
    pair {
      token0 {
        symbol
      }
      token1 {
        symbol
      }
      amount0In
      amount0Out
      amount1In
      amount1Out
      amountUSD
      to
    }
  }
}
```

Pair Daily Aggregated

Day data is useful for building charts and historical views around entities. To get stats about a pair in daily buckets query for day entities bounded by timestamps. This query gets the first 100 days after the given unix timestamp on the DAI/WETH pair.

```
{
  pairDayDatas(first: 100, orderBy: date, orderDirection: asc,
    where: {
      pairAddress: "0xa478c2975ablea89e8196811f51a7b7ade33eb11",
      date_gt: 1592505859
    }
  ) {
    date
    dailyVolumeToken0
    dailyVolumeToken1
    dailyVolumeUSD
    reserveUSD
  }
}
```

Token Data

Token data can be fetched using the token contract address as an ID. Token data is aggregated across all pairs the token is included in. Any token that is included in some pair in Uniswap can be queried.

Token Overview

Get a snapshot of the current stats on a token in Uniswap. This query fetches current stats on DAI. The allPairs field gets the first 200 pairs DAI is included in sorted by liquidity in derived USD.

```
{
  token(id: "0x6b175474e89094c44da98b954eedeac495271d0f") {
    name
    symbol
    decimals
    derivedETH
    tradeVolumeUSD
    totalLiquidity
  }
}
```

All Tokens in Uniswap

Similar to fetching all pairs (see above), you can query all tokens in Uniswap. Because The Graph service limits return size to 1000 entities use graphql skip query. (Note this query will not work in the graph sandbox and more resembles the structure of a query you'd pass to some graphql middleware like [Apollo](#)).

```
{
  query tokens($skip: Int!) {
    tokens(first: 1000, skip: $skip) {
      id
      name
      symbol
    }
  }
}
```

Token Transactions

To get transactions that include a token you'll need to first fetch an array of pairs that the token is included in (this can be done with the allPairs field on the Token entity.) Once you have an array of pairs the token is included in, filter on that in the transaction lookup.

This query fetches the latest 30 mints, swaps, and burns involving DAI. The allPairs array could look something like this where we include the DAI/WETH pair address and the DAI/USDC pair address.

```
allPairs = [
  "0xa478c2975ablea89e8196811f51a7b7ade33eb11",
  "0xae461ca67b15dc8dc81ce7615e0320da1a9ab8d5"
]

query($allPairs: [String!]) {
  mints(first: 30, where: { pair_in: $allPairs }, orderBy: timestamp, orderDirection: desc) {
    transaction {
      id
      timestamp
    }
  }
}
```

```

        to
        liquidity
        amount0
        amount1
        amountUSD
    }
burns(first: 30, where: { pair_in: $allPairs }, orderBy: timestamp, orderDirection: desc) {
    transaction {
        id
        timestamp
    }
    to
    liquidity
    amount0
    amount1
    amountUSD
}
swaps(first: 30, where: { pair_in: $allPairs }, orderBy: timestamp, orderDirection: desc) {
    transaction {
        id
        timestamp
    }
    amount0In
    amount0Out
    amount1In
    amount1Out
    amountUSD
    to
}
}

```

Token Daily Aggregated

Like pair and global daily lookups, tokens have daily entities that can be queries as well. This query gets daily information for DAI. Note that you may want to sort in ascending order to receive your days from oldest to most recent in the return array.

```

{
  tokenDayDatas(orderBy: date, orderDirection: asc,
  where: {
    token: "0x6b175474e89094c44da98b954eedeac495271d0f"
  }
) {
  id
  date
  priceUSD
  totalLiquidityToken
  totalLiquidityUSD
  totalLiquidityETH
  dailyVolumeETH
  dailyVolumeToken
  dailyVolumeUSD
}
}

```

ETH Price

You can use the Bundle entity to query current USD price of ETH in Uniswap based on a weighted average of stablecoins.

```

{
  bundle(id: "1" ) {
    ethPrice
  }
}

```

id: governance-reference title: Governance Reference

The updated reference for the newly deployed Governor Bravo is available via [Etherscan](#), some of the reference material below may be out of date.

The Uniswap protocol is governed and upgraded by UNI token holders, using three distinct components; the UNI token, governance module, and Timelock. Together, these contracts allow the community to propose, vote, and implement changes to the uniswap protocol.

Any addresses with more than 2.5M UNI delegated to it may propose governance actions, which contain finished, executable code. When a proposal is created, the community can cast their votes during a 3 day voting period. If a majority, and at least 4M votes are cast for the proposal, it is queued in the Timelock, and may be executed in a minimum of 2 days.

Timelock

The Timelock contract can modify system parameters, logic, and contracts in a 'time-delayed, opt-out' upgrade pattern. Timelock has a hard-coded minimum delay of 2 days, which is the least amount of notice possible for a governance action. Each proposed action will be published at a minimum of 2 days in the future from the time of announcement. Major upgrades, such as changing the risk system, may have up to a 30 day delay. Timelock is controlled by the governance module; pending and completed governance actions can be monitored on the Timelock Dashboard.

Key Events

DelegateChanged

```
DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate)
```

Emitted when an account changes its delegate.

DelegateVotesChanged

```
DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance)
```

Emitted when a delegate account's vote balance changes.

ProposalCreated

```
ProposalCreated(uint id, address proposer, address[] targets, uint[] values, string[] signatures, bytes[] calldatas, uint startBlock, uint endBlock, string description)
```

Emitted when a new proposal is created.

VoteCast

```
VoteCast(address voter, uint proposalId, bool support, uint votes)
```

Emitted when a vote has been cast on a proposal.

ProposalCanceled

```
ProposalCanceled(uint id)
```

Emitted when a proposal has been canceled.

ProposalQueued

```
ProposalQueued(uint id, uint eta)
```

Emitted when a proposal has been queued in the Timelock.

ProposalExecuted

```
ProposalExecuted(uint id)
```

Emitted when a proposal has been executed in the Timelock.

Read-Only Functions: UNI

Get Current Votes

```
function getCurrentVotes(address account) returns (uint96)
```

Returns the balance of votes for an account as of the current block.

| Name | Type | |
|---------|---------|--|
| account | address | Address of the account of which to retrieve the number of votes. |

Get Prior Votes

```
function getPriorVotes(address account, uint blockNumber) returns (uint96)
```

Returns the prior number of votes for an account at a specific block number. The block number passed must be a finalized block or the function will revert.

| Name | Type | |
|-------------|---------|--|
| account | address | Address of the account of which to retrieve the prior number of votes. |
| blocknumber | uint | The block number at which to retrieve the prior number of votes. |
| unnamed | uint96 | The number of prior votes |

State-Changing Functions: UNI

Delegate

```
function delegate(address delegatee)
```

Delegate votes from the sender to the delegatee. Users can delegate to 1 address at a time, and the number of votes added to the delegatee's vote count is equivalent to the balance of UNI in the user's account. Votes are delegated from the current block and onward, until the sender delegates again, or transfers their UNI.

| Name | Type | |
|-----------|---------|--|
| delegatee | address | The address to which msg.sender wishes to delegate their votes to. |

Delegate By Signature

```
function delegateBySig(address delegatee, uint nonce, uint expiry, uint8 v, bytes32 r, bytes32 s)
```

Delegate votes from the sender to the delegatee. Users can delegate to 1 address at a time, and the number of votes added to the delegatee's vote count is equivalent to the balance of UNI in the user's account. Votes are delegated from the current block and onward, until the sender delegates again, or transfers their UNI.

| Name | Type | |
|-----------|---------|---|
| delegatee | address | The address to which msg.sender wishes to delegate their vote to |
| nonce | uint | The contract state required to match the signature. This can be retrieved from the contract's public nonces mapping |
| expiry | uint | The time when the signature expires. A block timestamp in seconds since the unix epoch. |
| v | uint | The recovery byte of the signature. |
| r | bytes32 | Half of the ECDSA signature pair. |

| | | |
|---|---------|-----------------------------------|
| s | bytes32 | Half of the ECDSA signature pair. |
|---|---------|-----------------------------------|

Read-Only Functions: Governor Alpha

Quorum Votes

```
function quorumVotes() public pure returns (uint)
```

Returns the minimum number of votes required for a proposal to succeed.

Proposal Threshold

```
function proposalThreshold() returns (uint)
```

Returns the minimum number of votes required for an account to create a proposal.

Proposal Max Operations

```
function proposalMaxOperations() returns (uint)
```

Returns the maximum number of actions that can be included in a proposal. Actions are functions calls that will be made when a proposal succeeds and executes.

Voting Delay

```
function votingDelay() returns (uint)
```

Returns the number of blocks to wait before voting on a proposal may begin. This value is added to the current block number when a proposal is created.

Voting Period

```
function votingPeriod() returns (uint)
```

Returns the duration of voting on a proposal, in blocks.

Get Actions

```
function getActions(uint proposalId) returns (uint proposalId) public view returns (address[] memory targets,
uint[] memory values, string[] memory signatures, bytes[] memory calldatas)
```

Gets the actions of a selected proposal. Pass a proposal ID and get the targets, values, signatures and calldatas of that proposal.

| Name | Type | |
|------------|------|--------------------|
| proposalId | uint | ID of the proposal |

Returns:

- Array of addresses of contracts the proposal calls.
- Array of unsigned integers the proposal uses as values.
- Array of strings of the proposal's signatures.
- Array of calldata bytes of the proposal.

Get Receipt

```
function getReceipt(uint proposalId, address voter) returns (Receipt memory)
```

Returns a proposal ballot receipt of a given voter.

| Name | Type | |
|------------|---------|--|
| proposalId | uint | ID of the proposal in which to get a voter's ballot receipt. |
| voter | address | Address of the account of a proposal voter. |
| Receipt | struct | A Receipt struct for the ballot of the voter address. |

State

```
function state(uint proposalId) returns (ProposalState)
```

Returns enum of type ProposalState, possible types are: -Pending -Active -Canceled -Defeated -Succeeded -Queued -Expired -andExecuted

| Name | Type | |
|------------|------|--------------------|
| proposalId | uint | ID of the proposal |

State-Changing Functions: Governor Alpha

Propose

```
function propose(address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldatas, string memory description) returns (uint)
```

Creates a Proposal to change the protocol.

Proposals will be voted on by delegated voters. If there is sufficient support before the voting period ends, the proposal shall be automatically enacted. Enacted proposals are queued and executed in the Timelock contract.

The sender must hold more UNI than the current proposal threshold (proposalThreshold()) as of the immediately previous block. The proposal can have up to 10 actions (based on proposalMaxOperations()).

The proposer cannot create another proposal if they currently have a pending or active proposal. It is not possible to queue two identical actions in the same block (due to a restriction in the Timelock), therefore actions in a single proposal must be unique, and unique proposals that share an identical action must be queued in different blocks.

| Name | Type | |
|-------------|---------|---|
| targets | address | The ordered list of target addresses for calls to be made during proposal execution. This array must be the same length as all other array parameters in this function. |
| values | uint | The ordered list of values (i.e. msg.value) to be passed to the calls made during proposal execution. This array must be the same length as all other array parameters in this function |
| signatures | string | The ordered list of function signatures to be passed during execution. This array must be the same length as all other array parameters in this function. |
| calldatas | bytes | The ordered list of data to be passed to each individual function call during proposal execution. This array must be the same length as all other array parameters in this function. |
| description | string | A human readable description of the proposal and the changes it will enact. |
| Unnamed | uint | Returns ID of the new proposal |

Queue

```
function queue(uint proposalId)
```

After a proposal has succeeded, any address can call the queue method to move the proposal into the Timelock queue. A proposal can only be queued if it has succeeded.

| Name | Type | |
|------------|------|-----------------------------------|
| proposalId | uint | ID of a given successful proposal |

Execute

```
function execute(uint proposalId) payable
```

After the Timelock delay period, any account may invoke the execute method to apply the changes from the proposal to the target contracts. This will invoke each of the actions described in the proposal. This function is payable so the Timelock contract can invoke payable functions that were selected in the proposal.

| Name | Type | |
|------------|------|-----------------------------------|
| proposalId | uint | ID of a given successful proposal |

Cancel

```
function queue(uint proposalId)
```

Cancel a proposal that has not yet been executed. The Guardian is the only one who may execute this unless the proposer does not maintain the delegates required to create a proposal. If the proposer does not have more delegates than the proposal threshold, anyone can cancel the proposal.

| Name | Type | |
|------------|------|----------------------------|
| proposalId | uint | ID of a proposal to cancel |

Cast Vote

```
function castVote(uint proposalId, bool support)
```

Cast a vote on a proposal. The account's voting weight is determined by its number of delegated votes at the time the proposal becomes active.

| Name | Type | |
|------------|------|---|
| proposalId | uint | ID of a given successful proposal |
| support | bool | A boolean of true for 'yes' or false for 'no' on the proposal vote. |

Cast Vote By Signature

```
function castVoteBySig(uint proposalId, bool support, uint8 v, bytes32 r, bytes32 s)
```

Cast a vote on a proposal. The account's voting weight is determined by its number of delegated votes at the time the proposal became active. This method has the same purpose as Cast Vote, but instead enables offline signatures to participate in governance voting. For more details on how to create an offline signature, review EIP-712.

| Name | Type | |
|------------|---------|---|
| proposalId | uint | ID of a given successful proposal |
| support | bool | A boolean of true for 'yes' or false for 'no' on the proposal vote. |
| expiry | uint | The time when the signature expires. A block timestamp in seconds since the unix epoch. |
| v | uint | The recovery byte of the signature. |
| r | bytes32 | Half of the ECDSA signature pair. |
| s | bytes32 | Half of the ECDSA signature pair. |

id: factory title: Factory

Factory

Code

[UniswapV2Factory.sol](#)

Address

`UniswapV2Factory` is deployed at `0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f` on the Ethereum [mainnet](#), and the [Ropsten](#), [Rinkeby](#), [Görli](#), and [Kovan](#) testnets. It was built from commit [8160750](#).

Events

PairCreated

```
event PairCreated(address indexed token0, address indexed token1, address pair, uint);
```

Emitted each time a pair is created via [createPair](#).

- `token0` is guaranteed to be strictly less than `token1` by sort order.
- The final `uint` log value will be `1` for the first pair created, `2` for the second, etc. (see [allPairs/getPair](#)).

Read-Only Functions

getPair

```
function getPair(address tokenA, address tokenB) external view returns (address pair);
```

Returns the address of the pair for `tokenA` and `tokenB`, if it has been created, else `address(0)` (`0x000`).

- `tokenA` and `tokenB` are interchangeable.
- Pair addresses can also be calculated deterministically via the SDK.

allPairs

```
function allPairs(uint) external view returns (address pair);
```

Returns the address of the `n` th pair (`0` -indexed) created through the factory, or `address(0)` (`0x000`) if not enough pairs have been created yet.

- Pass `0` for the address of the first pair created, `1` for the second, etc.

allPairsLength

```
function allPairsLength() external view returns (uint);
```

Returns the total number of pairs created through the factory so far.

feeTo

```
function feeTo() external view returns (address);
```

See [Protocol Charge Calculation](#).

feeToSetter

```
function feeToSetter() external view returns (address);
```

The address allowed to change [feeTo](#).

State-Changing Functions

createPair

```
function createPair(address tokenA, address tokenB) external returns (address pair);
```

Creates a pair for `tokenA` and `tokenB` if one doesn't exist already.

- `tokenA` and `tokenB` are interchangeable.
- Emits [PairCreated](#).

Interface

```
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol';

pragma solidity >=0.5.0;

interface IUniswapV2Factory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function createPair(address tokenA, address tokenB) external returns (address pair);
}
```

ABI

```
import IUniswapV2Factory from '@uniswap/v2-core/build/IUniswapV2Factory.json'
```

<https://unpkg.com/@uniswap/v2-core@1.0.0/build/IUniswapV2Factory.json>

id: pair title: Pair

This documentation covers Uniswap-specific functionality. For ERC-20 functionality, see [Pair \(ERC-20\)](#).

Code

[UniswapV2Pair.sol](#)

Address

See [Pair Addresses](#).

Events

Mint

```
event Mint(address indexed sender, uint amount0, uint amount1);
```

Emitted each time liquidity tokens are created via [mint](#).

Burn

```
event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
```

Emitted each time liquidity tokens are destroyed via [burn](#).

Swap

```
event Swap(
    address indexed sender,
    uint amount0In,
    uint amount1In,
    uint amount0Out,
    uint amount1Out,
    address indexed to
);
```

Emitted each time a swap occurs via [swap](#).

Sync

```
event Sync(uint112 reserve0, uint112 reserve1);
```

Emitted each time reserves are updated via [mint](#), [burn](#), [swap](#), or [sync](#).

Read-Only Functions

MINIMUM_LIQUIDITY

```
function MINIMUM_LIQUIDITY() external pure returns (uint);
```

Returns `1000` for all pairs. See [Minimum Liquidity](#).

factory

```
function factory() external view returns (address);
```

Returns the [factory address](#).

token0

```
function token0() external view returns (address);
```

Returns the address of the pair token with the lower sort order.

token1

```
function token1() external view returns (address);
```

Returns the address of the pair token with the higher sort order.

getReserves

```
function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
```

Returns the reserves of token0 and token1 used to price trades and distribute liquidity. See [Pricing](#). Also returns the `block.timestamp` (mod `2**32`) of the last block during which an interaction occurred for the pair.

price0CumulativeLast

```
function price0CumulativeLast() external view returns (uint);
```

See [Oracles](#).

price1CumulativeLast

```
function price1CumulativeLast() external view returns (uint);
```

See [Oracles](#).

kLast

```
function kLast() external view returns (uint);
```

Returns the product of the reserves as of the most recent liquidity event. See [Protocol Charge Calculation](#).

State-Changing Functions

mint

```
function mint(address to) external returns (uint liquidity);
```

Creates pool tokens.

- Emits [Mint](#), [Sync](#), [Transfer](#).

burn

```
function burn(address to) external returns (uint amount0, uint amount1);
```

Destroys pool tokens.

- Emits [Burn](#), [Sync](#), [Transfer](#).

swap

```
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
```

Swaps tokens. For regular swaps, `data.length` must be `0`. Also see [Flash Swaps](#).

- Emits [Swap](#), [Sync](#).

skim

```
function skim(address to) external;
```

See the [whitepaper](#).

sync

```
function sync() external;
```

See the [whitepaper](#).

- Emits [Sync](#).

Interface

```
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol';
```

```

pragma solidity >=0.5.0;

interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s)
        external;

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;
}

```

ABI

```
import IUniswapV2Pair from '@uniswap/v2-core/build/IUniswapV2Pair.json'
```

<https://unpkg.com/@uniswap/v2-core@1.0.0/build/IUniswapV2Pair.json>

id: Pair-ERC-20 title: Pair (ERC-20)

This documentation covers ERC-20 functionality for denominating pool tokens. For Uniswap-specific functionality, see [Pair](#).

Code

[UniswapV2ERC20.sol](#)

Events

Approval

```
event Approval(address indexed owner, address indexed spender, uint value);
```

Emitted each time an approval occurs via [approve](#) or [permit](#).

Transfer

```
event Transfer(address indexed from, address indexed to, uint value);
```

Emitted each time a transfer occurs via [transfer](#), [transferFrom](#), [mint](#), or [burn](#).

Read-Only Functions

name

```
function name() external pure returns (string memory);
```

Returns `Uniswap V2` for all pairs.

symbol

```
function symbol() external pure returns (string memory);
```

Returns `UNI-V2` for all pairs.

decimals

```
function decimals() external pure returns (uint8);
```

Returns `18` for all pairs.

totalSupply

```
function totalSupply() external view returns (uint);
```

Returns the total amount of pool tokens for a pair.

balanceOf

```
function balanceOf(address owner) external view returns (uint);
```

Returns the amount of pool tokens owned by an address.

allowance

```
function allowance(address owner, address spender) external view returns (uint);
```

Returns the amount of liquidity tokens owned by an address that a spender is allowed to transfer via [transferFrom](#).

DOMAIN_SEPARATOR

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
```

Returns a domain separator for use in [permit](#).

PERMIT_TYPEHASH

```
function PERMIT_TYPEHASH() external view returns (bytes32);
```

Returns a typehash for use in [permit](#).

nonces

```
function nonces(address owner) external view returns (uint);
```

Returns the current nonce for an address for use in [permit](#).

State-Changing Functions

approve

```
function approve(address spender, uint value) external returns (bool);
```

Lets `msg.sender` set their allowance for a spender.

- Emits [Approval](#).

transfer

```
function transfer(address to, uint value) external returns (bool);
```

Lets `msg.sender` send pool tokens to an address.

- Emits [Transfer](#).

transferFrom

```
function transferFrom(address from, address to, uint value) external returns (bool);
```

Sends pool tokens from one address to another.

- Requires approval.
- Emits [Transfer](#).

permit

```
function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
```

Sets the allowance for a spender where approval is granted via a signature.

- See [Using Permit](#).
- Emits [Approval](#).

Interface

```
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2ERC20.sol';

pragma solidity >=0.5.0;

interface IUniswapV2ERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
```

```

function symbol() external pure returns (string memory);
function decimals() external pure returns (uint8);
function totalSupply() external view returns (uint);
function balanceOf(address owner) external view returns (uint);
function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint value) external returns (bool);
function transfer(address to, uint value) external returns (bool);
function transferFrom(address from, address to, uint value) external returns (bool);

function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function nonces(address owner) external view returns (uint);

function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s)
external;
}

```

ABI

```
import IUniswapV2ERC20 from '@uniswap/v2-core/build/IUniswapV2ERC20.json'
```

<https://unpkg.com/@uniswap/v2-core@1.0.0/build/IUniswapV2ERC20.json>

id: library title: Library

Library

Code

[UniswapV2Library.sol](#)

Internal Functions

sortTokens

```
function sortTokens(address tokenA, address tokenB) internal pure returns (address token0, address token1);
```

Sorts token addresses.

pairFor

```
function pairFor(address factory, address tokenA, address tokenB) internal pure returns (address pair);
```

Calculates the address for a pair without making any external calls via the v2 SDK.

getReserves

```
function getReserves(address factory, address tokenA, address tokenB) internal view returns (uint reserveA, uint reserveB);
```

Calls [getReserves](#) on the pair for the passed tokens, and returns the results sorted in the order that the parameters were passed in.

quote

```
function quote(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB);
```

Given some asset amount and reserves, returns an amount of the other asset representing equivalent value.

- Useful for calculating optimal token amounts before calling [mint](#).

getAmountOut

```
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint amountOut);
```

Given an *input* asset amount, returns the maximum *output* amount of the other asset (accounting for fees) given reserves.

- Used in [getAmountsOut](#).

getAmountIn

```
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) internal pure returns (uint amountIn);
```

Returns the minimum *input* asset amount required to buy the given *output* asset amount (accounting for fees) given reserves.

- Used in [getAmountsIn](#).

getAmountsOut

```
function getAmountsOut(uint amountIn, address[] memory path) internal view returns (uint[] memory amounts);
```

Given an *input* asset amount and an array of token addresses, calculates all subsequent maximum *output* token amounts by calling [getReserves](#) for each pair of token addresses in the path in turn, and using these to call [getAmountOut](#).

- Useful for calculating optimal token amounts before calling [swap](#).

getAmountsIn

```
function getAmountsIn(address factory, uint amountOut, address[] memory path) internal view returns (uint[] memory amounts);
```

Given an *output* asset amount and an array of token addresses, calculates all preceding minimum *input* token amounts by calling [getReserves](#) for each pair of token addresses in the path in turn, and using these to call [getAmountIn](#).

- Useful for calculating optimal token amounts before calling [swap](#).

id: router-01 title: Router01

UniswapV2Router01 should not be used any longer, because of the discovery of a [low severity bug](#) and the fact that some methods do not work with tokens that take fees on transfer. The current recommendation is to use [UniswapV2Router02](#).

Code

[UniswapV2Router01.sol](#)

Address

UniswapV2Router01 is deployed at `0xf164fC0Ec4E93095b804a4795bBe1e041497b92a` on the Ethereum [mainnet](#), and the [Ropsten](#), [Rinkeby](#), [Görli](#), and [Kovan](#) testnets. It was built from commit [2ad7da2](#).

Read-Only Functions

factory

```
function factory() external pure returns (address);
```

Returns [factory address](#).

WETH

```
function WETH() external pure returns (address);
```

Returns the [canonical WETH address](#) on the Ethereum [mainnet](#), or the [Ropsten](#), [Rinkeby](#), [Görli](#), or [Kovan](#) testnets.

State-Changing Functions

addLiquidity

```
function addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB, uint liquidity);
```

Adds liquidity to an ERC-20 \Rightarrow ERC-20 pool.

- To cover all possible scenarios, `msg.sender` should have already given the router an allowance of at least `amountADesired/amountBDesired` on `tokenA/tokenB`.
- Always adds assets at the ideal ratio, according to the price when the transaction is executed.
- If a pool for the passed tokens does not exists, one is created automatically, and exactly `amountADesired/amountBDesired` tokens are added.

| Name | Type | |
|----------------|---------|---|
| tokenA | address | A pool token. |
| tokenB | address | A pool token. |
| amountADesired | uint | The amount of <code>tokenA</code> to add as liquidity if the <code>B/A</code> price is \leq <code>amountBDesired/amountADesired</code> (<code>A</code> depreciates). |
| amountBDesired | uint | The amount of <code>tokenB</code> to add as liquidity if the <code>A/B</code> price is \leq <code>amountADesired/amountBDesired</code> (<code>B</code> depreciates). |
| amountAMin | uint | Bounds the extent to which the <code>B/A</code> price can go up before the transaction reverts. Must be \leq <code>amountADesired</code> . |
| amountBMin | uint | Bounds the extent to which the <code>A/B</code> price can go up before the transaction reverts. Must be \leq <code>amountBDesired</code> . |
| to | address | Recipient of the liquidity tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountA | uint | The amount of <code>tokenA</code> sent to the pool. |
| amountB | uint | The amount of <code>tokenB</code> sent to the pool. |
| liquidity | uint | The amount of liquidity tokens minted. |

addLiquidityETH

```
function addLiquidityETH(
    address token,
    uint amountTokenDesired,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external payable returns (uint amountToken, uint amountETH, uint liquidity);
```

Adds liquidity to an ERC-20 \Rightarrow WETH pool with ETH.

- To cover all possible scenarios, `msg.sender` should have already given the router an allowance of at least `amountTokenDesired` on token.
- Always adds assets at the ideal ratio, according to the price when the transaction is executed.
- `msg.value` is treated as a amountETHDesired.
- Leftover ETH, if any, is returned to `msg.sender`.
- If a pool for the passed token and WETH does not exists, one is created automatically, and exactly `amountTokenDesired / msg.value` tokens are added.

| Name | Type | |
|---------------------------------|---------|---|
| token | address | A pool token. |
| amountTokenDesired | uint | The amount of token to add as liquidity if the WETH/token price is $\leq \frac{msg.value}{amountTokenDesired}$ (token depreciates). |
| msg.value
(amountETHDesired) | uint | The amount of ETH to add as liquidity if the token/WETH price is $\leq \frac{amountTokenDesired}{msg.value}$ (WETH depreciates). |
| amountTokenMin | uint | Bounds the extent to which the WETH/token price can go up before the transaction reverts. Must be $\leq amountTokenDesired$. |
| amountETHMin | uint | Bounds the extent to which the token/WETH price can go up before the transaction reverts. Must be $\leq msg.value$. |
| to | address | Recipient of the liquidity tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountToken | uint | The amount of token sent to the pool. |
| amountETH | uint | The amount of ETH converted to WETH and sent to the pool. |
| liquidity | uint | The amount of liquidity tokens minted. |

removeLiquidity

```
function removeLiquidity(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB);
```

Removes liquidity from an ERC-20⇒ERC-20 pool.

- `msg.sender` should have already given the router an allowance of at least liquidity on the pool.

| Name | Type | |
|------------|---------|---|
| tokenA | address | A pool token. |
| tokenB | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountAMin | uint | The minimum amount of tokenA that must be received for the transaction not to revert. |
| amountBMin | uint | The minimum amount of tokenB that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountA | uint | The amount of tokenA received. |
| amountB | uint | The amount of tokenB received. |

removeLiquidityETH

```
function removeLiquidityETH(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external returns (uint amountToken, uint amountETH);
```

Removes liquidity from an ERC-20⇒WETH pool and receive ETH.

- `msg.sender` should have already given the router an allowance of at least liquidity on the pool.

| Name | Type | |
|----------------|---------|--|
| token | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountTokenMin | uint | The minimum amount of token that must be received for the transaction not to revert. |
| amountETHMin | uint | The minimum amount of ETH that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountToken | uint | The amount of token received. |
| amountETH | uint | The amount of ETH received. |

removeLiquidityWithPermit

```
function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountA, uint amountB);
```

Removes liquidity from an ERC-20⇒ERC-20 pool without pre-approval, thanks to [permit](#).

| Name | Type | |
|------------|---------|---|
| tokenA | address | A pool token. |
| tokenB | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountAMin | uint | The minimum amount of tokenA that must be received for the transaction not to revert. |
| amountBMin | uint | The minimum amount of tokenB that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| approveMax | bool | Whether or not the approval amount in the signature is for liquidity or <code>uint(-1)</code> . |
| v | uint8 | The v component of the permit signature. |
| r | bytes32 | The r component of the permit signature. |
| s | bytes32 | The s component of the permit signature. |

| | | |
|---------|------|--------------------------------|
| | | |
| amountA | uint | The amount of tokenA received. |
| amountB | uint | The amount of tokenB received. |

removeLiquidityETHWithPermit

```
function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountToken, uint amountETH);
```

Removes liquidity from an ERC-20⇒WETTH pool and receive ETH without pre-approval, thanks to [permit](#).

| Name | Type | |
|----------------|---------|---|
| token | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountTokenMin | uint | The minimum amount of token that must be received for the transaction not to revert. |
| amountETHMin | uint | The minimum amount of ETH that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| approveMax | bool | Whether or not the approval amount in the signature is for liquidity or <code>uint(-1)</code> . |
| v | uint8 | The v component of the permit signature. |
| r | bytes32 | The r component of the permit signature. |
| s | bytes32 | The s component of the permit signature. |
| amountToken | uint | The amount of token received. |
| amountETH | uint | The amount of ETH received. |

swapExactTokensForTokens

```
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
```

Swaps an exact amount of input tokens for as many output tokens as possible, along the route determined by the path. The first element of path is the input token, the last is the output token, and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- `msg.sender` should have already given the router an allowance of at least `amountIn` on the input token.

| Name | Type | |
|--------------|-----------------------|--|
| amountIn | uint | The amount of input tokens to send. |
| amountOutMin | uint | The minimum amount of output tokens that must be received for the transaction not to revert. |
| path | address[]
calldata | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |

| | | |
|-----------------------|----------------------------|---|
| <code>to</code> | <code>address</code> | Recipient of the output tokens. |
| <code>deadline</code> | <code>uint</code> | Unix timestamp after which the transaction will revert. |
| <code>amounts</code> | <code>uint[] memory</code> | The input token amount and all subsequent output token amounts. |

swapTokensForExactTokens

```
function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
```

Receive an exact amount of output tokens for as few input tokens as possible, along the route determined by the path. The first element of path is the input token, the last is the output token, and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- `msg.sender` should have already given the router an allowance of at least `amountInMax` on the input token.

| Name | Type | |
|--------------------------|---------------------------------|--|
| <code>amountOut</code> | <code>uint</code> | The amount of output tokens to receive. |
| <code>amountInMax</code> | <code>uint</code> | The maximum amount of input tokens that can be required before the transaction reverts. |
| <code>path</code> | <code>address[] calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| <code>to</code> | <code>address</code> | Recipient of the output tokens. |
| <code>deadline</code> | <code>uint</code> | Unix timestamp after which the transaction will revert. |
| <code>amounts</code> | <code>uint[] memory</code> | The input token amount and all subsequent output token amounts. |

swapExactETHForTokens

```
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
```

Swaps an exact amount of ETH for as many output tokens as possible, along the route determined by the path. The first element of path must be [WETH](#), the last is the output token, and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

| Name | Type | |
|---|---------------------------------|--|
| <code>msg.value</code>
<code>(amountIn)</code> | <code>uint</code> | The amount of ETH to send. |
| <code>amountOutMin</code> | <code>uint</code> | The minimum amount of output tokens that must be received for the transaction not to revert. |
| <code>path</code> | <code>address[] calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| <code>to</code> | <code>address</code> | Recipient of the output tokens. |
| <code>deadline</code> | <code>uint</code> | Unix timestamp after which the transaction will revert. |
| <code>amounts</code> | <code>uint[] memory</code> | The input token amount and all subsequent output token amounts. |

swapTokensForExactETH

```

function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
    external
    returns (uint[] memory amounts);

```

Receive an exact amount of ETH for as few input tokens as possible, along the route determined by the path. The first element of path is the input token, the last must be [WETH](#), and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- `msg.sender` should have already given the router an allowance of at least `amountInMax` on the input token.
- If the `to` address is a smart contract, it must have the ability to receive ETH.

| Name | Type | |
|--------------------------|---------------------------------|--|
| <code>amountOut</code> | <code>uint</code> | The amount of ETH to receive. |
| <code>amountInMax</code> | <code>uint</code> | The maximum amount of input tokens that can be required before the transaction reverts. |
| <code>path</code> | <code>address[] calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| <code>to</code> | <code>address</code> | Recipient of ETH. |
| <code>deadline</code> | <code>uint</code> | Unix timestamp after which the transaction will revert. |
| <code>amounts</code> | <code>uint[] memory</code> | The input token amount and all subsequent output token amounts. |

swapExactTokensForETH

```

function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    returns (uint[] memory amounts);

```

Swaps an exact amount of tokens for as much ETH as possible, along the route determined by the path. The first element of path is the input token, the last must be [WETH](#), and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- If the `to` address is a smart contract, it must have the ability to receive ETH.

| Name | Type | |
|---------------------------|---------------------------------|--|
| <code>amountIn</code> | <code>uint</code> | The amount of input tokens to send. |
| <code>amountOutMin</code> | <code>uint</code> | The minimum amount of output tokens that must be received for the transaction not to revert. |
| <code>path</code> | <code>address[] calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| <code>to</code> | <code>address</code> | Recipient of the ETH. |
| <code>deadline</code> | <code>uint</code> | Unix timestamp after which the transaction will revert. |
| <code>amounts</code> | <code>uint[] memory</code> | The input token amount and all subsequent output token amounts. |

swapETHForExactTokens

```

function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);

```

Receive an exact amount of tokens for as little ETH as possible, along the route determined by the path. The first element of path must be [WETH](#), the last is the output token and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- Leftover ETH, if any, is returned to `msg.sender`.

| Name | Type | |
|------|------|--|
| | | |

| | | |
|----------------------------|-----------------------|--|
| amountOut | uint | The amount of tokens to receive. |
| msg.value
(amountInMax) | uint | The maximum amount of ETH that can be required before the transaction reverts. |
| path | address[]
calldata | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | address | Recipient of the output tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amounts | uint[] memory | The input token amount and all subsequent output token amounts. |

quote

See [quote](#).

getAmountOut

See [getAmountOut](#).

getAmountIn

This function contains a low severity bug, do not use.

getAmountsOut

```
function getAmountsOut(uint amountIn, address[] memory path) public view returns (uint[] memory amounts);
```

See [getAmountsOut](#).

getAmountsIn

```
function getAmountsIn(uint amountOut, address[] memory path) public view returns (uint[] memory amounts);
```

See [getAmountsIn](#).

Interface

```
import '@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol';
```

```
pragma solidity >=0.6.2;

interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
```

```

        uint deadline
) external payable returns (uint amountToken, uint amountETH, uint liquidity);
function removeLiquidity(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB);
function removeLiquidityETH(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external returns (uint amountToken, uint amountETH);
function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountA, uint amountB);
function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountToken, uint amountETH);
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);

function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);

```

```
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint amountOut);
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint amountIn);
function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);
function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory amounts);
}
```

ABI

```
import IUniswapV2Router01 from '@uniswap/v2-periphery/build/IUniswapV2Router01.json'
```

<https://unpkg.com/@uniswap/v2-periphery@1.0.0-beta.0/build/IUniswapV2Router01.json>

id: router-02 title: Router02

Because routers are stateless and do not hold token balances, they can be replaced safely and trustlessly, if necessary. This may happen if more efficient smart contract patterns are discovered, or if additional functionality is desired. For this reason, routers have *release numbers*, starting at 01 . This is currently recommended release, 02 .

Code

[UniswapV2Router02.sol](#)

Address

UniswapV2Router02 is deployed at 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D on the Ethereum [mainnet](#), and the [Ropsten](#), [Rinkeby](#), [Görli](#), and [Kovan](#) testnets. It was built from commit [6961711](#).

Read-Only Functions

factory

```
function factory() external pure returns (address);
```

Returns [factory address](#).

WETH

```
function WETH() external pure returns (address);
```

Returns the [canonical WETH address](#) on the Ethereum [mainnet](#), or the [Ropsten](#), [Rinkeby](#), [Görli](#), or [Kovan](#) testnets.

quote

See [quote](#).

getAmountOut

See [getAmountOut](#).

getAmountIn

See [getAmountIn](#).

getAmountsOut

```
function getAmountsOut(uint amountIn, address[] memory path) public view returns (uint[] memory amounts);
```

See [getAmountsOut](#).

getAmountsIn

```
function getAmountsIn(uint amountOut, address[] memory path) public view returns (uint[] memory amounts);
```

See [getAmountsIn](#).

State-Changing Functions

addLiquidity

```
function addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB, uint liquidity);
```

Adds liquidity to an ERC-20 \Rightarrow ERC-20 pool.

- To cover all possible scenarios, `msg.sender` should have already given the router an allowance of at least `amountADesired/amountBDesired` on `tokenA/tokenB`.
- Always adds assets at the ideal ratio, according to the price when the transaction is executed.
- If a pool for the passed tokens does not exists, one is created automatically, and exactly `amountADesired/amountBDesired` tokens are added.

| Name | Type | |
|----------------|---------|---|
| tokenA | address | A pool token. |
| tokenB | address | A pool token. |
| amountADesired | uint | The amount of tokenA to add as liquidity if the B/A price is \leq <code>amountBDesired/amountADesired</code> (A depreciates). |
| amountBDesired | uint | The amount of tokenB to add as liquidity if the A/B price is \leq <code>amountADesired/amountBDesired</code> (B depreciates). |
| amountAMin | uint | Bounds the extent to which the B/A price can go up before the transaction reverts. Must be \leq <code>amountADesired</code> . |
| amountBMin | uint | Bounds the extent to which the A/B price can go up before the transaction reverts. Must be \leq <code>amountBDesired</code> . |
| to | address | Recipient of the liquidity tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountA | uint | The amount of tokenA sent to the pool. |
| amountB | uint | The amount of tokenB sent to the pool. |
| liquidity | uint | The amount of liquidity tokens minted. |

addLiquidityETH

```
function addLiquidityETH(
    address token,
    uint amountTokenDesired,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
```

```

    uint deadline
) external payable returns (uint amountToken, uint amountETH, uint liquidity);

```

Adds liquidity to an ERC-20⇒WETH pool with ETH.

- To cover all possible scenarios, `msg.sender` should have already given the router an allowance of at least `amountTokenDesired` on token.
- Always adds assets at the ideal ratio, according to the price when the transaction is executed.
- `msg.value` is treated as a `amountETHDesired`.
- Leftover ETH, if any, is returned to `msg.sender`.
- If a pool for the passed token and WETH does not exists, one is created automatically, and exactly `amountTokenDesired`/`msg.value` tokens are added.

| Name | Type | |
|---------------------------------|---------|---|
| token | address | A pool token. |
| amountTokenDesired | uint | The amount of token to add as liquidity if the WETH/token price is $\leq \text{msg.value}/\text{amountTokenDesired}$ (token depreciates). |
| msg.value
(amountETHDesired) | uint | The amount of ETH to add as liquidity if the token/WETH price is $\leq \text{amountTokenDesired}/\text{msg.value}$ (WETH depreciates). |
| amountTokenMin | uint | Bounds the extent to which the WETH/token price can go up before the transaction reverts. Must be $\leq \text{amountTokenDesired}$. |
| amountETHMin | uint | Bounds the extent to which the token/WETH price can go up before the transaction reverts. Must be $\leq \text{msg.value}$. |
| to | address | Recipient of the liquidity tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountToken | uint | The amount of token sent to the pool. |
| amountETH | uint | The amount of ETH converted to WETH and sent to the pool. |
| liquidity | uint | The amount of liquidity tokens minted. |

removeLiquidity

```

function removeLiquidity(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB);

```

Removes liquidity from an ERC-20⇒ERC-20 pool.

- `msg.sender` should have already given the router an allowance of at least liquidity on the pool.

| Name | Type | |
|------------|---------|---|
| tokenA | address | A pool token. |
| tokenB | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountAMin | uint | The minimum amount of tokenA that must be received for the transaction not to revert. |
| amountBMin | uint | The minimum amount of tokenB that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountA | uint | The amount of tokenA received. |

| | | |
|---------|------|--------------------------------|
| amountB | uint | The amount of tokenB received. |
|---------|------|--------------------------------|

removeLiquidityETH

```
function removeLiquidityETH(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external returns (uint amountToken, uint amountETH);
```

Removes liquidity from an ERC-20⇒WETH pool and receive ETH.

- `msg.sender` should have already given the router an allowance of at least liquidity on the pool.

| Name | Type | |
|----------------|---------|--|
| token | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountTokenMin | uint | The minimum amount of token that must be received for the transaction not to revert. |
| amountETHMin | uint | The minimum amount of ETH that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountToken | uint | The amount of token received. |
| amountETH | uint | The amount of ETH received. |

removeLiquidityWithPermit

```
function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountA, uint amountB);
```

Removes liquidity from an ERC-20⇒ERC-20 pool without pre-approval, thanks to [permit](#).

| Name | Type | |
|------------|---------|---|
| tokenA | address | A pool token. |
| tokenB | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountAMin | uint | The minimum amount of tokenA that must be received for the transaction not to revert. |
| amountBMin | uint | The minimum amount of tokenB that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| approveMax | bool | Whether or not the approval amount in the signature is for liquidity or <code>uint(-1)</code> . |
| v | uint8 | The v component of the permit signature. |

| | | |
|---------|---------|--|
| r | bytes32 | The r component of the permit signature. |
| s | bytes32 | The s component of the permit signature. |
| amountA | uint | The amount of tokenA received. |
| amountB | uint | The amount of tokenB received. |

removeLiquidityETHWithPermit

```
function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountToken, uint amountETH);
```

Removes liquidity from an ERC-20==WETTH pool and receive ETH without pre-approval, thanks to [permit](#).

| Name | Type | |
|----------------|---------|---|
| token | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountTokenMin | uint | The minimum amount of token that must be received for the transaction not to revert. |
| amountETHMin | uint | The minimum amount of ETH that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| approveMax | bool | Whether or not the approval amount in the signature is for liquidity or <code>uint(-1)</code> . |
| v | uint8 | The v component of the permit signature. |
| r | bytes32 | The r component of the permit signature. |
| s | bytes32 | The s component of the permit signature. |
| amountToken | uint | The amount of token received. |
| amountETH | uint | The amount of ETH received. |

removeLiquidityETHSupportingFeeOnTransferTokens

```
function removeLiquidityETHSupportingFeeOnTransferTokens(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external returns (uint amountETH);
```

Identical to [removeLiquidityETH](#), but succeeds for tokens that take a fee on transfer.

- `msg.sender` should have already given the router an allowance of at least liquidity on the pool.

| Name | Type | |
|----------------|---------|--|
| token | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountTokenMin | uint | The minimum amount of token that must be received for the transaction not to revert. |

| | | |
|--------------|---------|--|
| amountETHMin | uint | The minimum amount of ETH that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amountETH | uint | The amount of ETH received. |

removeLiquidityETHWithPermitSupportingFeeOnTransferTokens

```
function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountETH);
```

Identical to [removeLiquidityETHWithPermit](#), but succeeds for tokens that take a fee on transfer.

| Name | Type | |
|----------------|---------|---|
| token | address | A pool token. |
| liquidity | uint | The amount of liquidity tokens to remove. |
| amountTokenMin | uint | The minimum amount of token that must be received for the transaction not to revert. |
| amountETHMin | uint | The minimum amount of ETH that must be received for the transaction not to revert. |
| to | address | Recipient of the underlying assets. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| approveMax | bool | Whether or not the approval amount in the signature is for liquidity or <code>uint(-1)</code> . |
| v | uint8 | The v component of the permit signature. |
| r | bytes32 | The r component of the permit signature. |
| s | bytes32 | The s component of the permit signature. |
| amountETH | uint | The amount of ETH received. |

swapExactTokensForTokens

```
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
```

Swaps an exact amount of input tokens for as many output tokens as possible, along the route determined by the path. The first element of path is the input token, the last is the output token, and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- `msg.sender` should have already given the router an allowance of at least `amountIn` on the input token.

| Name | Type | |
|--------------|------|--|
| amountIn | uint | The amount of input tokens to send. |
| amountOutMin | uint | The minimum amount of output tokens that must be received for the transaction not to revert. |

| | | |
|----------|--------------------|--|
| path | address[] calldata | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | address | Recipient of the output tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amounts | uint[] memory | The input token amount and all subsequent output token amounts. |

swapTokensForExactTokens

```
function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
```

Receive an exact amount of output tokens for as few input tokens as possible, along the route determined by the path. The first element of path is the input token, the last is the output token, and any intermediate elements represent intermediate tokens to trade through (if, for example, a direct pair does not exist).

- `msg.sender` should have already given the router an allowance of at least `amountInMax` on the input token.

| Name | Type | |
|-------------|--------------------|--|
| amountOut | uint | The amount of output tokens to receive. |
| amountInMax | uint | The maximum amount of input tokens that can be required before the transaction reverts. |
| path | address[] calldata | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | address | Recipient of the output tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amounts | uint[] memory | The input token amount and all subsequent output token amounts. |

swapExactETHForTokens

```
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
```

Swaps an exact amount of ETH for as many output tokens as possible, along the route determined by the path. The first element of path must be [WETH](#), the last is the output token, and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

| Name | Type | |
|--|--------------------|--|
| <code>msg.value</code> (<code>amountIn</code>) | uint | The amount of ETH to send. |
| amountOutMin | uint | The minimum amount of output tokens that must be received for the transaction not to revert. |
| path | address[] calldata | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | address | Recipient of the output tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amounts | uint[] memory | The input token amount and all subsequent output token amounts. |

swapTokensForExactETH

```
function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
    external
    returns (uint[] memory amounts);
```

Receive an exact amount of ETH for as few input tokens as possible, along the route determined by the path. The first element of path is the input token, the last must be [WETH](#), and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- `msg.sender` should have already given the router an allowance of at least `amountInMax` on the input token.
- If the `to` address is a smart contract, it must have the ability to receive ETH.

| Name | Type | |
|-------------|--------------------|--|
| amountOut | uint | The amount of ETH to receive. |
| amountInMax | uint | The maximum amount of input tokens that can be required before the transaction reverts. |
| path | address[] calldata | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | address | Recipient of ETH. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amounts | uint[] memory | The input token amount and all subsequent output token amounts. |

swapExactTokensForETH

```
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    returns (uint[] memory amounts);
```

Swaps an exact amount of tokens for as much ETH as possible, along the route determined by the path. The first element of path is the input token, the last must be [WETH](#), and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- If the `to` address is a smart contract, it must have the ability to receive ETH.

| Name | Type | |
|--------------|--------------------|--|
| amountIn | uint | The amount of input tokens to send. |
| amountOutMin | uint | The minimum amount of output tokens that must be received for the transaction not to revert. |
| path | address[] calldata | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | address | Recipient of the ETH. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| amounts | uint[] memory | The input token amount and all subsequent output token amounts. |

swapETHForExactTokens

```
function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
```

Receive an exact amount of tokens for as little ETH as possible, along the route determined by the path. The first element of path must be [WETH](#), the last is the output token and any intermediate elements represent intermediate pairs to trade through (if, for example, a direct pair does not exist).

- Leftover ETH, if any, is returned to `msg.sender`.

| Name | Type | |
|--|---|--|
| <code>amountOut</code> | <code>uint</code> | The amount of tokens to receive. |
| <code>msg.value</code>
(<code>amountInMax</code>) | <code>uint</code> | The maximum amount of ETH that can be required before the transaction reverts. |
| <code>path</code> | <code>address[]</code>
<code>calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| <code>to</code> | <code>address</code> | Recipient of the output tokens. |
| <code>deadline</code> | <code>uint</code> | Unix timestamp after which the transaction will revert. |
| <code>amounts</code> | <code>uint[] memory</code> | The input token amount and all subsequent output token amounts. |

swapExactTokensForTokensSupportingFeeOnTransferTokens

```
function swapExactTokensForTokensSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external;
```

Identical to [swapExactTokensForTokens](#), but succeeds for tokens that take a fee on transfer.

- `msg.sender` should have already given the router an allowance of at least `amountIn` on the input token.

| Name | Type | |
|---------------------------|---|--|
| <code>amountIn</code> | <code>uint</code> | The amount of input tokens to send. |
| <code>amountOutMin</code> | <code>uint</code> | The minimum amount of output tokens that must be received for the transaction not to revert. |
| <code>path</code> | <code>address[]</code>
<code>calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| <code>to</code> | <code>address</code> | Recipient of the output tokens. |
| <code>deadline</code> | <code>uint</code> | Unix timestamp after which the transaction will revert. |

swapExactETHForTokensSupportingFeeOnTransferTokens

```
function swapExactETHForTokensSupportingFeeOnTransferTokens(
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external payable;
```

Identical to [swapExactETHForTokens](#), but succeeds for tokens that take a fee on transfer.

| Name | Type | |
|---|---|--|
| <code>msg.value</code>
(<code>amountIn</code>) | <code>uint</code> | The amount of ETH to send. |
| <code>amountOutMin</code> | <code>uint</code> | The minimum amount of output tokens that must be received for the transaction not to revert. |
| <code>path</code> | <code>address[]</code>
<code>calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| <code>to</code> | <code>address</code> | Recipient of the output tokens. |
| <code>deadline</code> | <code>uint</code> | Unix timestamp after which the transaction will revert. |

swapExactTokensForETHSupportingFeeOnTransferTokens

```
function swapExactTokensForETHSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external;
```

Identical to [swapExactTokensForETH](#), but succeeds for tokens that take a fee on transfer.

- If the to address is a smart contract, it must have the ability to receive ETH.

| Name | Type | |
|--------------|-----------------------|--|
| amountIn | uint | The amount of input tokens to send. |
| amountOutMin | uint | The minimum amount of output tokens that must be received for the transaction not to revert. |
| path | address[]
calldata | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | address | Recipient of the ETH. |
| deadline | uint | Unix timestamp after which the transaction will revert. |

Interface

```
import '@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol';

pragma solidity >=0.6.2;

interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
```

```

        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);
function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountA, uint amountB);
function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountToken, uint amountETH);
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);

function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint amountOut);
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint amountIn);
function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);
function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory amounts);
}

interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    )
}

```

```

) external returns (uint amountETH);
function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountETH);

function swapExactTokensForTokensSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external;
function swapExactETHForTokensSupportingFeeOnTransferTokens(
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external payable;
function swapExactTokensForETHSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external;
}

```

ABI

```
import IUniswapV2Router02 from '@uniswap/v2-periphery/build/IUniswapV2Router02.json'
```

<https://unpkg.com/@uniswap/v2-periphery@1.1.0-beta.0/build/IUniswapV2Router02.json>

id: common-errors title: Common Errors

This document covers a few error codes frequently encountered while building on Uniswap V2.

UniswapV2: K

This is an error that is frequently encountered, and requires a bit of context to understand it.

The Uniswap constant product formula is " $X * Y = K$ ". Where X and Y represent the respective reserve balances of two ERC-20 tokens, and "K" represents the product of the reserves. It is this "K" to which the "K" error refers.

In essence, the "K" error means that a trade was attempted that somehow left the trading pair with less reserves than should be there, and as a result the transaction is reverted.

This can have a few different causes.

Fee On Transfer Tokens

The most common examples are caused by "fee on transfer" tokens.

Inclusive Fee On Transfer Tokens

In most cases, a fee on transfer token burns or diverts a small portion of every transfer such that the recipient of the transfer ends up with slightly less than the sender gave. This is called an "inclusive" fee on transfer.

In the case of inclusive fee on transfer tokens, you can use the corresponding swap functions in the router contract which end with "[SupportingFeeOnTransfer](#)". These functions succeed by adjusting the "amountOutMin" parameter to check the recipient amount rather than the sending amount when calculating the invariant.

Exclusive Fee On Transfer Tokens

The other type, "exclusive" fee on transfer tokens, work by sending an additional transfer from the sending address after the primary transfer. Because the router contract cannot anticipate this trailing transfer when calculating the invariant, the transaction will either revert, or partially succeed by sending the primary transfer but breaking the pool upon the trailing transfer.

In the case of exclusive fee on transfer tokens, the SupportingFeeOnTransfer functions may work, but there will be some tokens designed in such a way that they fundamentally break the router. If you are still getting a "K" error when using these functions, you may need to make a fork of the router contract that accommodates your token design.

Rebasing Tokens

The less common instance of the "K" error is as a result of rebasing tokens.

Rebasing tokens can alter the balance of any addresses holding their tokens arbitrarily. This usually works at pre specified intervals and as a result of a handful of variables used in the economics of a rebasing token.

Rebasing tokens typically work in two ways.

Negative Rebasing Tokens

A negative rebasing token, the more common variant, deflates the balances of token owners. Because the rebasing is not triggered by transfers, the router cannot expect when or how a rebasing will happen. Once it does, the pair reserves will be unbalanced, and the next person to transact with the pair will bear the cost of the delta as a result of the rebasing.

Needless to say, an unenviable position.

Negative rebasing tokens have solved this error by altering their token contract to call [sync](#) on the trading pair at the end of every transaction involving the Uniswap router contract. Those interested in forking the router contract should anticipate that negative rebasing tokens will break the pair until the token contracts are updated to accommodate your new router.

Positive Rebasing Tokens

Positive rebasing tokens arbitrarily increase the balances of token holders. When a positive rebase happens, it creates a surplus that is unaccounted for in the trading pair. Because the extra tokens are unaccounted for in the trading pair, anyone can call [skim\(\)](#) on the trading pair and effectively steal the positive difference from the rebalance.

While positive rebalancing does not break any functionality of Uniswap, those interested in them should be aware that the positive balance found in any pair will be freely available for taking.

A Note on Rebasing Tokens

For those interested in building a rebasing token, a word of caution: many contracts involving decentralized trading and liquidity provisioning will break upon interacting with your token. An example approach that will lead to much easier integration in future protocols can be found in [CHAI](#). CHAI uses a wrapper function that contains the rebalancing within the wrapper, such that the redeemable token can be easily integrated into many different systems.

UniswapV2: LOCKED

The LOCKED error is a guard built into the router contract that prevents customized reentrancy contracts from attempting to return malicious code into the router contract at the end of a transaction.

This error is commonly encountered when using Ganache CLI to fork the Ethereum mainnet to a local instance as a part of a development environment. The error is a bug in Ganache-Cli that will hopefully be fixed in a future release by the truffle team.

A temporary fix is available by simply restarting the local fork.

No Access To Archive Node

This is an error with either Metamask or Ganache-CLI. It usually occurs after a local fork is instantiated and contracts are deployed but there is one failed transaction.

A temporary fix is available by restarting the local fork and resetting metamask.

UniswapV2: TRANSFER_FAILED

This means the core contract was unable to send tokens to the recipient. This is most likely due to a scam token, where the token owner has maliciously disabled the transfer function in a way that allows users to buy the token, but not sell them.

UniswapV2: EXPIRED

This is a result of a transaction that took too long to be broadcast to the mainnet.

Uniswap does not set gas prices natively, so most users default to the suggested gas prices in metamask. Sometimes metamask gets it wrong, though, and sets the gas price too low. If a swap takes more than 20 minutes to execute, the core contract won't allow it to go through.

Action Requires an Active Reserve

VM Exception While Processing Transaction: Action Requires an Active Reserve

This is potentially a ganache bug encountered when working on flash swaps. We haven't figured out the source of it yet.

Unable To Approve Transaction On The Front End

There are rare circumstances where users are unable to approve a token on the Uniswap front end.

This is a result of some token contracts taking steps to defend against malicious contracts that attempt to front run approvals and steal a user's tokens. It happens only when the user is trying to increase an approval allowance from a preallocated amount to a larger one, and only happens with a few token contracts.

The solution is have the user manually set the router contract approval amount to zero, then to the number they want. The easiest way to do this is through Etherscan.

id: inheritance-constructors title: Getting Started sidebar_position: 1

In this guide, we will write a smart contract that calls `flash` on a V3 pool and swaps the full amount withdrawn of `token0` and `token1` in the corresponding pools with the same token pair - but different fee tiers. After the swap, the contract will pay back the first pool and transfer profits to the original calling address.

Flash Transactions Overview

Flash transactions are an approach to transferring tokens on Ethereum that transfer token balances *before* the necessary conditions are met for those balances to be transferred. In the context of a swap, this would mean the output is sent from the swap before the input is received.

Uniswap V3 introduces a new function, `flash`, within the Pool contract. `Flash` withdraws a specified amount of both `token0` and `token1` to the `recipient` address. The withdrawn amount, plus the swap fees, will be due to the pool at the end of the transaction. `flash` includes a fourth parameter, `data`, which allows the caller to `abi.encode` any necessary data to be passed through the function and decoded later.

```
function flash(
    address recipient,
    uint256 amount0,
    uint256 amount1,
    bytes calldata data
) external override lock noDelegateCall {
```

The Flash Callback

`flash` will withdraw the tokens, but how are they paid back? To understand this, we must look inside the `flash` function code. midway through the [flash](#) function, we see this:

```
IUniswapV3FlashCallback(msg.sender).uniswapV3FlashCallback(fee0, fee1, data);
```

This step calls the `FlashCallback` function on `msg.sender` - which passes the fee data needed to calculate the balances due to the pool, as well as any data encoded into the `data` parameter.

In V3 there are three separate callback functions, `uniswapV3SwapCallback`, `uniswapV3MintCallback`, and `uniswapV3FlashCallback`, each available to be overridden with custom logic. To write our arbitrage contract, we'll be calling `flash` and overriding the `uniswapV3FlashCallback` with the steps needed to finish executing our transaction.

Inheriting The V3 Contracts

Inherit `IUniswapV3FlashCallback` and `PeripheryPayments`, as we will use each in our program. Note these two inherited contracts already extend many other contracts that we will be using, such as [LowGasSafeMath](#) which we [attach](#) to types `uint256` and `int256`.

```
contract PairFlash is IUniswapV3FlashCallback, PeripheryPayments {
    using LowGasSafeMath for uint256;
```

```
using LowGasSafeMath for int256;
```

Declare an immutable public variable `swapRouter` of type `ISwapRouter`:

```
ISwapRouter public immutable swapRouter;
```

Declare the constructor here, which is executed once when the contract is deployed. Our constructor hardcodes the address of the V3 router, factory, and the address of weth9, the [ERC-20 wrapper](#) for ether.

```
constructor(
    ISwapRouter _swapRouter,
    address _factory,
    address _WETH9
) PeripheryImmutableState(_factory, _WETH9) {
    swapRouter = _swapRouter;
}
```

The full import section and contract declaration:

```
pragma solidity =0.7.6;
pragma abicoder v2;

import '@uniswap/v3-core/contracts/interfaces/callback/IUniswapV3FlashCallback.sol';
import '@uniswap/v3-core/contracts/libraries/LowGasSafeMath.sol';

import '@uniswap/v3-periphery/contracts/base/PeripheryPayments.sol';
import '@uniswap/v3-periphery/contracts/base/PeripheryImmutableState.sol';
import '@uniswap/v3-periphery/contracts/libraries/PoolAddress.sol';
import '@uniswap/v3-periphery/contracts/libraries/CallbackValidation.sol';
import '@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol';
import '@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol';

contract PairFlash is IUniswapV3FlashCallback, PeripheryPayments {
    using LowGasSafeMath for uint256;
    using LowGasSafeMath for int256;

    ISwapRouter public immutable swapRouter;

    constructor(
        ISwapRouter _swapRouter,
        address _factory,
        address _WETH9
    ) PeripheryImmutableState(_factory, _WETH9) {
        swapRouter = _swapRouter;
    }
}
```

id: calling-flash title: Calling Flash sidebar_position: 2

Parameter Structs

In order to call `flash`, we will need the flash parameters for the initial call, as well as any parameters we want to pass through to the callback.

The `FlashParams` struct will contain the token addresses and amounts we wish to pull out of the pool, as well as the three fee tiers used to determine which pool we are withdrawing from, and which we will be swapping with.

```
struct FlashParams {
    address token0;
    address token1;
    uint24 fee1;
    uint256 amount0;
    uint256 amount1;
    uint24 fee2;
    uint24 fee3;
}
```

The `FlashCallbackData` struct will contain the data we want to send to the callback. This includes `poolKey`, which expresses the sorted tokens with the matched fee tier, returned by the [PoolAddress](#) library.

```
struct FlashCallbackData {
    uint256 amount0;
    uint256 amount1;
    address payer;
    PoolAddress.PoolKey poolKey;
    uint24 poolFee2;
    uint24 poolFee3;
}
```

Pool Key

Now we'll start our function by assigning the relevant parameters from the `Flashparams` (which we have declared in memory as `params`) to our variable `poolKey`.

```
function initFlash(FlashParams memory params) external {
    PoolAddress.PoolKey memory poolKey =
        PoolAddress.PoolKey({token0: params.token0, token1: params.token1, fee: params.fee1});
}
```

Next we will declare `pool` as type [\[IUniswapV3Pool\]](#), which allows us to call `flash` on our desired pool contract.

```
IUniswapV3Pool pool = IUniswapV3Pool(PoolAddress.computeAddress(factory, poolKey));
```

Calling Flash

Finally, we call `flash` on our previously declared `pool`. In the last parameter, we `abi.encode` the `FlashCallbackData`, which will be decoded in the callback and used to inform the next steps of the transaction.

```
pool.flash(
    address(this),
    params.amount0,
    params.amount1,
    abi.encode(
        FlashCallbackData({
            amount0: params.amount0,
            amount1: params.amount1,
            payer: msg.sender,
            poolKey: poolKey,
            poolFee2: params.fee2,
            poolFee3: params.fee3
        })
    )
);
```

The full function:

```
//fee1 is the fee of the pool from the initial borrow
//fee2 is the fee of the first pool to arb from
//fee3 is the fee of the second pool to arb from
struct FlashParams {
    address token0;
    address token1;
    uint24 fee1;
    uint256 amount0;
    uint256 amount1;
    uint24 fee2;
    uint24 fee3;
}

// fee2 and fee3 are the two other fees associated with the two other pools of token0 and token1
struct FlashCallbackData {
    uint256 amount0;
    uint256 amount1;
```

```

        address payer;
        PoolAddress.PoolKey poolKey;
        uint24 poolFee2;
        uint24 poolFee3;
    }

function initFlash(FlashParams memory params) external {
    PoolAddress.PoolKey memory poolKey =
        PoolAddress.PoolKey((token0: params.token0, token1: params.token1, fee: params.fee));
    IUniswapV3Pool pool = IUniswapV3Pool(PoolAddress.computeAddress(factory, poolKey));
    pool.flash(
        address(this),
        params.amount0,
        params.amount1,
        abi.encode(
            FlashCallbackData({
                amount0: params.amount0,
                amount1: params.amount1,
                payer: msg.sender,
                poolKey: poolKey,
                poolFee2: params.fee2,
                poolFee3: params.fee3
            })
        )
    );
}

```

id: final-contract title: The Final Contract sidebar_position: 4

The Full Contract

```

// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity =0.7.6;
pragma abicoder v2;

import '@uniswap/v3-core/contracts/interfaces/callback/IUniswapV3FlashCallback.sol';
import '@uniswap/v3-core/contracts/libraries/LowGasSafeMath.sol';

import '@uniswap/v3-periphery/contracts/base/PeripheryPayments.sol';
import '@uniswap/v3-periphery/contracts/base/PeripheryImmutableState.sol';
import '@uniswap/v3-periphery/contracts/libraries/PoolAddress.sol';
import '@uniswap/v3-periphery/contracts/libraries/CallbackValidation.sol';
import '@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol';
import '@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol';

/// @title Flash contract implementation
/// @notice An example contract using the Uniswap V3 flash function
contract PairFlash is IUniswapV3FlashCallback, PeripheryImmutableState, PeripheryPayments {
    using LowGasSafeMath for uint256;
    using LowGasSafeMath for int256;

    ISwapRouter public immutable swapRouter;

    constructor(
        ISwapRouter _swapRouter,
        address _factory,
        address _WETH9
    ) PeripheryImmutableState(_factory, _WETH9) {
        swapRouter = _swapRouter;
    }

    /// @param fee0 The fee from calling flash for token0
    /// @param fee1 The fee from calling flash for token1
    /// @param data The data needed in the callback passed as FlashCallbackData from `initFlash`
    /// @notice implements the callback called from flash
    /// @dev fails if the flash is not profitable, meaning the amountOut from the flash is less than the amount
    borrowed
    function uniswapV3FlashCallback(

```

```

        uint256 fee0,
        uint256 fee1,
        bytes calldata data
    ) external override {
        FlashCallbackData memory decoded = abi.decode(data, (FlashCallbackData));
        CallbackValidation.verifyCallback(factory, decoded.poolKey);

        address token0 = decoded.poolKey.token0;
        address token1 = decoded.poolKey.token1;

        TransferHelper.safeApprove(token0, address(swapRouter), decoded.amount0);
        TransferHelper.safeApprove(token1, address(swapRouter), decoded.amount1);

        // profitable check
        // exactInputSingle will fail if this amount not met
        uint256 amount1Min = LowGasSafeMath.add(decoded.amount1, fee1);
        uint256 amount0Min = LowGasSafeMath.add(decoded.amount0, fee0);

        // call exactInputSingle for swapping token1 for token0 in pool w/fee2
        uint256 amountOut0 =
            swapRouter.exactInputSingle(
                ISwapRouter.ExactInputSingleParams({
                    tokenIn: token1,
                    tokenOut: token0,
                    fee: decoded.poolFee2,
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: decoded.amount1,
                    amountOutMinimum: amount0Min,
                    sqrtPriceLimitX96: 0
                })
            );
    };

    // call exactInputSingle for swapping token0 for token 1 in pool w/fee3
    uint256 amountOut1 =
        swapRouter.exactInputSingle(
            ISwapRouter.ExactInputSingleParams({
                tokenIn: token0,
                tokenOut: token1,
                fee: decoded.poolFee3,
                recipient: address(this),
                deadline: block.timestamp,
                amountIn: decoded.amount0,
                amountOutMinimum: amount1Min,
                sqrtPriceLimitX96: 0
            })
        );
    };

    // end up with amountOut0 of token0 from first swap and amountOut1 of token1 from second swap
    uint256 amount0Owed = LowGasSafeMath.add(decoded.amount0, fee0);
    uint256 amount1Owed = LowGasSafeMath.add(decoded.amount1, fee1);

    TransferHelper.safeApprove(token0, address(this), amount0Owed);
    TransferHelper.safeApprove(token1, address(this), amount1Owed);

    if (amount0Owed > 0) pay(token0, address(this), msg.sender, amount0Owed);
    if (amount1Owed > 0) pay(token1, address(this), msg.sender, amount1Owed);

    // if profitable pay profits to payer
    if (amountOut0 > amount0Owed) {
        uint256 profit0 = LowGasSafeMath.sub(amountOut0, amount0Owed);

        TransferHelper.safeApprove(token0, address(this), profit0);
        pay(token0, address(this), decoded.payer, profit0);
    }
    if (amountOut1 > amount1Owed) {
        uint256 profit1 = LowGasSafeMath.sub(amountOut1, amount1Owed);
        TransferHelper.safeApprove(token0, address(this), profit1);
        pay(token1, address(this), decoded.payer, profit1);
    }
}

```

```

}

//fee1 is the fee of the pool from the initial borrow
//fee2 is the fee of the first pool to arb from
//fee3 is the fee of the second pool to arb from
struct FlashParams {
    address token0;
    address token1;
    uint24 fee1;
    uint256 amount0;
    uint256 amount1;
    uint24 fee2;
    uint24 fee3;
}
// fee2 and fee3 are the two other fees associated with the two other pools of token0 and token1
struct FlashCallbackData {
    uint256 amount0;
    uint256 amount1;
    address payer;
    PoolAddress.PoolKey poolKey;
    uint24 poolFee2;
    uint24 poolFee3;
}

/// @param params The parameters necessary for flash and the callback, passed in as FlashParams
/// @notice Calls the pools flash function with data needed in `uniswapV3FlashCallback`
function initFlash(FlashParams memory params) external {
    PoolAddress.PoolKey memory poolKey =
        PoolAddress.PoolKey((token0: params.token0, token1: params.token1, fee: params.fee1));
    IUniswapV3Pool pool = IUniswapV3Pool(PoolAddress.computeAddress(factory, poolKey));
    // recipient of borrowed amounts
    // amount of token0 requested to borrow
    // amount of token1 requested to borrow
    // need amount 0 and amount1 in callback to pay back pool
    // recipient of flash should be THIS contract
    pool.flash(
        address(this),
        params.amount0,
        params.amount1,
        abi.encode(
            FlashCallbackData({
                amount0: params.amount0,
                amount1: params.amount1,
                payer: msg.sender,
                poolKey: poolKey,
                poolFee2: params.fee2,
                poolFee3: params.fee3
            })
        )
    );
}
}
}

```

id: flash-callback title: The Flash Callback sidebar_position: 3

Setting Up The Callback

Here we will override the flash callback with our custom logic to execute the desired swaps and pay the profits to the original `msg.sender`.

Declare the `uniswapV3FlashCallback` function and override it.

```

function uniswapV3FlashCallback(
    uint256 fee0,
    uint256 fee1,
    bytes calldata data
) external override {

```

Declare a variable `decoded` in memory and assign it to the [decoded data](#) previously encoded into the calldata.

```
FlashCallbackData memory decoded = abi.decode(data, (FlashCallbackData));
```

Each callback must be validated to verify that the call originated from a genuine V3 pool. Otherwise, the pool contract would be vulnerable to attack via an EOA manipulating the callback function.

```
CallbackValidation.verifyCallback(factory, decoded.poolKey);
```

Assign local variables of type `address` as `token0` and `token1` to approve the router to interact with the tokens from the flash.

```
address token0 = decoded.poolKey.token0;
address token1 = decoded.poolKey.token1;

TransferHelper.safeApprove(token0, address(swapRouter), decoded.amount0);
TransferHelper.safeApprove(token1, address(swapRouter), decoded.amount1);
```

Code in a minimum amount out for both of the upcoming swaps, such that the following swaps will revert if we do not receive a profitable trade.

```
uint256 amount1Min = LowGasSafeMath.add(decoded.amount1, fee1);
uint256 amount0Min = LowGasSafeMath.add(decoded.amount0, fee0);
```

Initiating A Swap

Call the first of two swaps, calling `exactInputSingle` on the [router interface](#) contract. In this call, we are using the previously declared `amount0In` as the minimum amount out, and assigning the returned balance of the swap to `amountOut0`.

Most of These function arguments have already been discussed, except for two new introductions:

`sqrtPriceLimitX96` : This value limits the price that the swap can change the pool to. Remember that price is always expressed in the pool contract as `token1` in terms of `token0`. This is useful for circumstances where the user wants to swap *up until* a specific price. For this example, we will set it to 0, which makes the argument inactive.

`deadline` : this is the timestamp after which the transaction will revert, to protect the transaction from dramatic changes in price environment that can happen if the transaction is pending for too long. For this example, we will set it far in the future for the sake of simplicity.

The first swap takes the `amount1` that we withdrew from the original pool, and passes that amount as the input amount for a single swap that trades a fixed input for the maximum amount of possible output. It calls this function on the pool determined by our previous token pair, but with the next fee tier in our list of three.

```
uint256 amountOut0 =
    swapRouter.exactInputSingle(
        ISwapRouter.ExactInputSingleParams({
            tokenIn: token1,
            tokenOut: token0,
            fee: decoded.poolFee2,
            recipient: address(this),
            deadline: block.timestamp + 200,
            amountIn: decoded.amount1,
            amountOutMinimum: amount0Min,
            sqrtPriceLimitX96: 0
        })
    );
```

Populate the second of two swaps, this time with the last fee tier and with the `amount0` that we withdrew from the original pool.

```
uint256 amountOut1 =
    swapRouter.exactInputSingle(
        ISwapRouter.ExactInputSingleParams({
            tokenIn: token0,
            tokenOut: token1,
            fee: decoded.poolFee3,
            recipient: address(this),
            deadline: block.timestamp + 200,
            amountIn: decoded.amount0,
            amountOutMinimum: amount1Min,
            sqrtPriceLimitX96: 0
        })
    );
```

```
)  
);
```

Paying back the pool

To pay the original pool back for the flash transaction, first calculate the balance due to it and approve the router to transfer the tokens in our contract back to the pool.

```
uint256 amount0Owed = LowGasSafeMath.add(decoded.amount0, fee0);  
uint256 amount1Owed = LowGasSafeMath.add(decoded.amount1, fee1);  
  
TransferHelper.safeApprove(token0, address(this), amount0Owed);  
TransferHelper.safeApprove(token1, address(this), amount1Owed);
```

If there is any balance due to the token, use simple logic to call `pay`. Remember that the callback function is being called by the pool itself, which is why we can call `pay` despite the function being marked `internal`.

```
if (amount0Owed > 0) pay(token0, address(this), msg.sender, amount0Owed);  
if (amount1Owed > 0) pay(token1, address(this), msg.sender, amount1Owed);
```

Send the profits to the `payer` : the original `msg.sender` of the `initFlash` function, which executed the flash transaction and in turn triggered the callback.

```
if (amountOut0 > amount0Owed) {  
    uint256 profit0 = LowGasSafeMath.sub(amountOut0, amount0Owed);  
  
    TransferHelper.safeApprove(token0, address(this), profit0);  
    pay(token0, address(this), decoded.payer, profit0);  
}  
  
if (amountOut1 > amount1Owed) {  
    uint256 profit1 = LowGasSafeMath.sub(amountOut1, amount1Owed);  
    TransferHelper.safeApprove(token0, address(this), profit1);  
    pay(token1, address(this), decoded.payer, profit1);  
}
```

The full function

```
function uniswapV3FlashCallback(  
    uint256 fee0,  
    uint256 fee1,  
    bytes calldata data  
) external override {  
    FlashCallbackData memory decoded = abi.decode(data, (FlashCallbackData));  
    CallbackValidation.verifyCallback(factory, decoded.poolKey);  
  
    address token0 = decoded.poolKey.token0;  
    address token1 = decoded.poolKey.token1;  
  
    TransferHelper.safeApprove(token0, address(swapRouter), decoded.amount0);  
    TransferHelper.safeApprove(token1, address(swapRouter), decoded.amount1);  
  
    // profitable check  
    // exactInputSingle will fail if this amount not met  
    uint256 amount1Min = LowGasSafeMath.add(decoded.amount1, fee1);  
    uint256 amount0Min = LowGasSafeMath.add(decoded.amount0, fee0);  
  
    // call exactInputSingle for swapping token1 for token0 in pool w/fee2  
    uint256 amountOut0 =  
        swapRouter.exactInputSingle(  
            ISwapRouter.ExactInputSingleParams({  
                tokenIn: token1,  
                tokenOut: token0,  
                fee: decoded.poolFee2,  
                recipient: address(this),
```

```

        deadline: block.timestamp + 200,
        amountIn: decoded.amount1,
        amountOutMinimum: amount0Min,
        sqrtPriceLimitX96: 0
    })
);

// call exactInputSingle for swapping token0 for token 1 in pool w/fee3
uint256 amountOut1 =
    swapRouter.exactInputSingle(
        ISwapRouter.ExactInputSingleParams({
            tokenIn: token0,
            tokenOut: token1,
            fee: decoded.poolFee3,
            recipient: address(this),
            deadline: block.timestamp + 200,
            amountIn: decoded.amount0,
            amountOutMinimum: amount1Min,
            sqrtPriceLimitX96: 0
        })
);
;

// end up with amountOut0 of token0 from first swap and amountOut1 of token1 from second swap
uint256 amount0Owed = LowGasSafeMath.add(decoded.amount0, fee0);
uint256 amount1Owed = LowGasSafeMath.add(decoded.amount1, fee1);

TransferHelper.safeApprove(token0, address(this), amount0Owed);
TransferHelper.safeApprove(token1, address(this), amount1Owed);

if (amount0Owed > 0) pay(token0, address(this), msg.sender, amount0Owed);
if (amount1Owed > 0) pay(token1, address(this), msg.sender, amount1Owed);

// if profitable pay profits to payer
if (amountOut0 > amount0Owed) {
    uint256 profit0 = LowGasSafeMath.sub(amountOut0, amount0Owed);

    TransferHelper.safeApprove(token0, address(this), profit0);
    pay(token0, address(this), decoded.payer, profit0);
}

if (amountOut1 > amount1Owed) {
    uint256 profit1 = LowGasSafeMath.sub(amountOut1, amount1Owed);
    TransferHelper.safeApprove(token0, address(this), profit1);
    pay(token1, address(this), decoded.payer, profit1);
}
}
}

```

id: license-modifications title: License Modifications

Licensing

Please note that Uniswap V3 is under [BUSL license](#) until the Change Date, currently 2023-04-01. Exceptions to the license may be specified by Uniswap Governance via Additional Use Grants, which can, for example, allow V3 to be deployed on new chains. Please follow the [Uniswap Governance process](#) to request a DAO vote for exceptions to the license, or to move up the Change Date.

License changes must be enacted via the [ENS domain](#) uniswap.eth, which is controlled by Uniswap Governance. This means (among other things) that Governance has the power to associate arbitrary text with any subdomain of the form X.uniswap.eth. Modifications of the Change Date should be specified at v3-core-license-date.uniswap.eth, and Additional Use Grants should be specified at v3-core-license-grants.uniswap.eth. The process for associating text with a subdomain is detailed below:

- ▶ ENS Subdomain Details & Process

Proposals

Proposals are submitted via `GovernorBravoDelegator @ 0x408ED6354d4973f66138C91495F2f2FCbd8724C3`, a proxy contract currently pointing to the implementation at `0x53a328F4086d7C0F1Fa19e594c9b842125263026`. NPM packages for consuming the governance contract ABIs, and details on previous versions, are available [here](#)

- ▶ Governor Bravo #propose Parameters

Populating Proposal Calldata

Below is an example of using a scripting environment to generate a proposal. This is for educational purposes only – that example assumes access to a private key with a sufficient amount of delegated UNI to submit a proposal, which is an insecure practice. There are several ways to generate a proposal transaction and submit it to Ethereum; this example should only be used for reference and not in production.

- ▶ Populating `Propose` Calldata

Helpful Links

- [Governor Bravo Proxy](#)
- [Governor Bravo Delegate](#)
- [ENS Subnode Record Update Details](#)

id: overview title: Overview sidebar_position: 1

Introduction

As a DeFi project, token creator, or other interested party, you may want to *incentivize in-range liquidity provision* on a Uniswap V3 pool. This guide describes one particular incentivization scheme at a high level, as implemented in [uniswap-v3-staker](#).

The Setting

Let's start by defining some terms. We refer to programs which incentivize liquidity as `Incentive`s; they're characterized by the following parameters:

- `rewardToken` : Perhaps the most important parameter, would-be incentivizers must pick the ERC20 token which they would like to distribute as a reward for providing liquidity.
- `pool` : The address of the Uniswap V3 pool in which liquidity must be provided.
- `startTime` : The UNIX timestamp at which rewards start to be distributed.
- `endTime` : The UNIX timestamp at which rewards start to decay.
- `refundee` : The address which has the right to reclaim any leftover rewards after the `Incentive` has concluded.

Finally, every `Incentive` has an associated `reward`, the total amount of `rewardToken`s that are allocated to be distributed over the lifecycle of the program.

Reward Math

Now that we have an idea of what an `Incentive` looks like, let's explore how rewards are actually allocated to participants. The next section will touch on the participation mechanics, so for now let's abstract this away and just focus on the high-level design.

Recall that `Incentive` creators pick a `reward` amount and a program duration. This directly corresponds to picking *an amount of rewardToken s to distribute per second*; let's call this the reward rate. So, for every second between `startTime` and `endTime`, a constant amount of tokens are distributed proportionally *among all in-range liquidity at that second*. Crucially, this counts *all* liquidity, not just liquidity that opts in to participating in the program. So, incentive creators should pick a reward rate that they deem worthwhile to distribute across (potentially) all in-range LPs for the duration of the program.

Staking

So, how do users participate in these programs? Note that this section requires a basic understanding of [how Uniswap V3 position NFTs work](#)

The first action a user must take in order to begin participating in an `Incentive` is to *deposit* their position NFT into the [canonical staking contract address](#), effectively temporarily giving custody over their NFT to this contract. This is necessary because, as we'll see later on, the staking contract needs to be able to guarantee that liquidity cannot be removed from NFTs participating in the program.

Once deposited, a user may then *stake* their NFT into any number of active `Incentive`s for the Uniswap V3 pool their NFT is tied to (note that this can happen atomically with an initial *deposit*). Staked NFTs then immediately start to earn rewards, according to the algorithm outlined above. Users may periodically claim accrued `rewardToken`s while the program is ongoing, or wait to claim until the program has concluded to minimize overhead.

Program Conclusion

There are two conditions that must be met for a program to be considered concluded:

1. `block.timestamp >= endTime` : In other words, the program's duration must have expired. However, this doesn't mark the official end of the program, as some users may still be participating right up until this `endTime` boundary and beyond, to maximize their rewards. This leads directly to the second condition.
2. All NFTs must be unstaked: A program can conclude only when every NFT which participated in it is unstaked. To ensure this is always possible, after the `endTime` of a program anyone may unstake *any* NFT (though of course they may not claim outstanding `rewardToken`s due to the NFT owner). This ensures that even if all users do not unstake themselves, someone can unstake them manually so that the program can end.

It's important that most or all programs fully conclude, primarily so that the `refuntee` can reclaim any unallocated rewards. What are the conditions under which unallocated rewards will remain? Well, recall that the reward rate is the same across *all* in-range liquidity. However, only program participants may actually claim accrued tokens, so it's likely that all programs will end up with a balance of `rewardToken`s that cannot be claimed. So, `refuntee`s will typically be incentivized to bring programs to an official conclusion. This slightly cumbersome design is a consequence of the difficulty of consistently allocating rewards proportional to Uniswap V3 liquidity.

A final note: stakers who remain in the program after `endTime` may actually see their rewards marginally augmented or (more likely) gradually diluted. The magnitude of these changes depend on stakers' share of the total active liquidity, the time spent staked after `endTime`, and the sequence of unstaking. In the worst case, rewards decay proportionally to the duration. For example, at 2x the duration, $\frac{1}{2}$ of rewards could remain, at 3x, $\frac{1}{3}$ could remain, etc. While somewhat complex, this behavior can largely be ignored from a game-theoretic standpoint. Stakers should simply attempt to unstake and claim rewards as soon as possible after `endTime`, an outcome that is likely in any case, as `refuntee`s will be eager to reclaim leftover rewards, and mass unstake stragglers.

id: collect-fees title: Collecting Fees sidebar_position: 3

Collect Fees

- Make sure to go through the [first guide](#) before continuing to this section.
- For each of these liquidity interaction examples, our contract must be in possession of the liquidity position NFT. Therefore, in any example where the NFT deposit is not coded into a function, the contract is assumed to already be in possession of it.

To collect the fees of an owner position, transfer the NFT from the calling address, assign the relevant variables from the NFT to local variables within our function, and pass those variables to the `nonfungiblePositionManager` to call `collect`.

This function collects all fees, sending them to the original owner of the NFT, while maintaining custody of the position NFT.

```
/// @notice Collects the fees associated with provided liquidity
/// @dev The contract must hold the erc721 token before it can collect fees
/// @param tokenId The id of the erc721 token
/// @return amount0 The amount of fees collected in token0
/// @return amount1 The amount of fees collected in token1
function collectAllFees(uint256 tokenId) external returns (uint256 amount0, uint256 amount1) {
    // Caller must own the ERC721 position
    // Call to safeTransfer will trigger `onERC721Received` which must return the selector else transfer will
fail
    nonfungiblePositionManager.safeTransferFrom(msg.sender, address(this), tokenId);

    // set amount0Max and amount1Max to uint256.max to collect all fees
    // alternatively can set recipient to msg.sender and avoid another transaction in `sendToOwner`
    INonfungiblePositionManager.CollectParams memory params =
        INonfungiblePositionManager.CollectParams({
            tokenId: tokenId,
            recipient: address(this),
            amount0Max: type(uint128).max,
            amount1Max: type(uint128).max
        });

    (amount0, amount1) = nonfungiblePositionManager.collect(params);

    // send collected feed back to owner
    _sendToOwner(tokenId, amount0, amount1);
}
```

Sending Fees To The Calling Address

This internal helper function sends any tokens, in the form of fees or position tokens, to the owner of an NFT.

In `_sendToOwner`, we pass the amount of fees due, previously populated in the last function, as arguments to `safeTransfer`, which transfers the fees to `owner`.

```

/// @notice Transfers funds to owner of NFT
/// @param tokenId The id of the erc721
/// @param amount0 The amount of token0
/// @param amount1 The amount of token1
function _sendToOwner(
    uint256 tokenId,
    uint256 amount0,
    uint256 amount1
) internal {
    // get owner of contract
    address owner = deposits[tokenId].owner;

    address token0 = deposits[tokenId].token0;
    address token1 = deposits[tokenId].token1;
    // send collected fees to owner
    TransferHelper.safeTransfer(token0, owner, amount0);
    TransferHelper.safeTransfer(token1, owner, amount1);
}

```

id: decrease-liquidity title: Decrease Liquidity sidebar_position: 4

Make sure to go through the [Setting Up Your Contract](#) before continuing to this section

Here we decrease the liquidity of our position without withdrawing all of it.

- This example assumes the contract already has possession of the position NFT, and requires the calling address to be the same address that deposited the position NFT to our contract.
- In production, `amount0Min` and `amount1Min` should be adjusted to create slippage protections.

Decrease Liquidity

```

/// @notice A function that decreases the current liquidity by half. An example to show how to call the
`decreaseLiquidity` function defined in periphery.
/// @param tokenId The id of the erc721 token
/// @return amount0 The amount received back in token0
/// @return amount1 The amount returned back in token1
function decreaseLiquidityInHalf(uint256 tokenId) external returns (uint256 amount0, uint256 amount1) {
    // caller must be the owner of the NFT
    require(msg.sender == deposits[tokenId].owner, 'Not the owner');
    // get liquidity data for tokenId
    uint128 liquidity = deposits[tokenId].liquidity;
    uint128 halfLiquidity = liquidity / 2;

    // amount0Min and amount1Min are price slippage checks
    // if the amount received after burning is not greater than these minimums, transaction will fail
    INonfungiblePositionManager.DecreaseLiquidityParams memory params =
        INonfungiblePositionManager.DecreaseLiquidityParams({
            tokenId: tokenId,
            liquidity: halfLiquidity,
            amount0Min: 0,
            amount1Min: 0,
            deadline: block.timestamp
        });

    (amount0, amount1) = nonfungiblePositionManager.decreaseLiquidity(params);

    //send liquidity back to owner
    _sendToOwner(tokenId, amount0, amount1);
}

```

Sending Fees To The Calling Address

This internal helper function sends any tokens, in the form of fees or position tokens, to the owner of an NFT.

In `_sendToOwner`, we pass the amount of fees due, previously populated in the last function, as arguments to `safeTransfer`, which transfers the fees to `owner`.

```

/// @notice Transfers funds to owner of NFT
/// @param tokenId The id of the erc721
/// @param amount0 The amount of token0
/// @param amount1 The amount of token1
function _sendToOwner(
    uint256 tokenId,
    uint256 amount0,
    uint256 amount1
) internal {
    // get owner of contract
    address owner = deposits[tokenId].owner;

    address token0 = deposits[tokenId].token0;
    address token1 = deposits[tokenId].token1;
    // send collected fees to owner
    TransferHelper.safeTransfer(token0, owner, amount0);
    TransferHelper.safeTransfer(token1, owner, amount1);
}

```

id: increase-liquidity title: Increase Liquidity sidebar_position: 5

Increase Liquidity Within The Current Range

Make sure to go through the [first guide](#) before continuing to this section

- This example assumes the contract already has custody of the NFT.
- We cannot change the boundaries of a given liquidity position using the Uniswap v3 protocol; `increaseLiquidity` can only increase the liquidity of a position.
- In production, `amount0Min` and `amount1Min` should be adjusted to create slippage protections.

```

/// @notice Increases liquidity in the current range
/// @dev Pool must be initialized already to add liquidity
/// @param tokenId The id of the erc721 token
/// @param amount0 The amount to add of token0
/// @param amount1 The amount to add of token1
function increaseLiquidityCurrentRange(
    uint256 tokenId,
    uint256 amountAdd0,
    uint256 amountAdd1
)
external
returns (
    uint128 liquidity,
    uint256 amount0,
    uint256 amount1
)
{
    INonfungiblePositionManager.IncreaseLiquidityParams memory params =
        INonfungiblePositionManager.IncreaseLiquidityParams({
            tokenId: tokenId,
            amount0Desired: amountAdd0,
            amount1Desired: amountAdd1,
            amount0Min: 0,
            amount1Min: 0,
            deadline: block.timestamp
        });
    (liquidity, amount0, amount1) = nonfungiblePositionManager.increaseLiquidity(params);
}

```

id: mint-a-position title: Mint a New Position sidebar_position: 2

Input Parameters

To mint a new position, we use the `nonFungiblePositionManager` and call `mint`.

For the sake of this example, we're hard coding the token amounts to be minted. In production, this would be a user-configurable function argument.

```
/// @notice Calls the mint function defined in periphery, mints the same amount of each token. For this example  
we are providing 1000 DAI and 1000 USDC in liquidity  
/// @return tokenId The id of the newly minted ERC721  
/// @return liquidity The amount of liquidity for the position  
/// @return amount0 The amount of token0  
/// @return amount1 The amount of token1  
function mintNewPosition()  
    external  
    returns (  
        uint256 tokenId,  
        uint128 liquidity,  
        uint256 amount0,  
        uint256 amount1  
    )  
{  
    // For this example, we will provide equal amounts of liquidity in both assets.  
    // Providing liquidity in both assets means liquidity will be earning fees and is considered in-range.  
    uint256 amount0ToMint = 1000;  
    uint256 amount1ToMint = 1000;
```

Calling Mint

Here we approve the `nonfungiblePositionManager` to use the contracts' tokens, then populate the `MintParams` struct and assign it to a local variable `params` that will be passed to the `nonfungiblePositionManager` when we call `mint`.

- By using `TickMath.MIN_TICK` and `TickMath.MAX_TICK`, we are providing liquidity across the whole range of the pool. In production you may want to specify a more concentrated position.
- We set `amount0Min` and `amount1Min` to zero for the example - but this would be a vulnerability in production. A function calling `mint` with no slippage protection would be vulnerable to a frontrunning attack designed to execute the `mint` call at an inaccurate price.
- For a more secure practice the developer would need to implement a slippage estimation process.
- Note that this function will not initialize a pool where one does not yet exist.

```
// Approve the position manager  
TransferHelper.safeApprove(DAI, address(nonfungiblePositionManager), amount0ToMint);  
TransferHelper.safeApprove(USDC, address(nonfungiblePositionManager), amount1ToMint);  
  
INonfungiblePositionManager.MintParams memory params =  
    INonfungiblePositionManager.MintParams({  
        token0: DAI,  
        token1: USDC,  
        fee: poolFee,  
        tickLower: TickMath.MIN_TICK,  
        tickUpper: TickMath.MAX_TICK,  
        amount0Desired: amount0ToMint,  
        amount1Desired: amount1ToMint,  
        amount0Min: 0,  
        amount1Min: 0,  
        recipient: address(this),  
        deadline: block.timestamp  
    });  
  
// Note that the pool defined by DAI/USDC and fee tier 0.3% must already be created and initialized in  
order to mint  
(tokenId, liquidity, amount0, amount1) = nonfungiblePositionManager.mint(params);
```

Updating The Deposit Mapping And Refunding The Calling Address

Now we can call the internal function we previously wrote in [Setting Up Your Contract](#). After that, we can take any liquidity leftover from minting and refund it to `msg.sender`.

```

    // Create a deposit
    _createDeposit(msg.sender, tokenId);

    // Remove allowance and refund in both assets.
    if (amount0 < amount0ToMint) {
        TransferHelper.safeApprove(DAI, address(nonfungiblePositionManager), 0);
        uint256 refund0 = amount0ToMint - amount0;
        TransferHelper.safeTransfer(DAI, msg.sender, refund0);
    }

    if (amount1 < amount1ToMint) {
        TransferHelper.safeApprove(USDC, address(nonfungiblePositionManager), 0);
        uint256 refund1 = amount1ToMint - amount1;
        TransferHelper.safeTransfer(USDC, msg.sender, refund1);
    }
}

```

The Full Example

```

/// @notice Calls the mint function defined in periphery, mints the same amount of each token. For this example
we are providing 1000 DAI and 1000 USDC in liquidity
/// @return tokenId The id of the newly minted ERC721
/// @return liquidity The amount of liquidity for the position
/// @return amount0 The amount of token0
/// @return amount1 The amount of token1
function mintNewPosition()
    external
    returns (
        uint256 tokenId,
        uint128 liquidity,
        uint256 amount0,
        uint256 amount1
    )
{
    // For this example, we will provide equal amounts of liquidity in both assets.
    // Providing liquidity in both assets means liquidity will be earning fees and is considered in-range.
    uint256 amount0ToMint = 1000;
    uint256 amount1ToMint = 1000;

    // Approve the position manager
    TransferHelper.safeApprove(DAI, address(nonfungiblePositionManager), amount0ToMint);
    TransferHelper.safeApprove(USDC, address(nonfungiblePositionManager), amount1ToMint);

    INonfungiblePositionManager.MintParams memory params =
        INonfungiblePositionManager.MintParams({
            token0: DAI,
            token1: USDC,
            fee: poolFee,
            tickLower: TickMath.MIN_TICK,
            tickUpper: TickMath.MAX_TICK,
            amount0Desired: amount0ToMint,
            amount1Desired: amount1ToMint,
            amount0Min: 0,
            amount1Min: 0,
            recipient: address(this),
            deadline: block.timestamp
        });

    // Note that the pool defined by DAI/USDC and fee tier 0.3% must already be created and initialized in
order to mint
    (tokenId, liquidity, amount0, amount1) = nonfungiblePositionManager.mint(params);

    // Create a deposit
    _createDeposit(msg.sender, tokenId);

    // Remove allowance and refund in both assets.
    if (amount0 < amount0ToMint) {
        TransferHelper.safeApprove(DAI, address(nonfungiblePositionManager), 0);

```

```

        uint256 refund0 = amount0ToMint - amount0;
        TransferHelper.safeTransfer(DAI, msg.sender, refund0);
    }

    if (amount1 < amount1ToMint) {
        TransferHelper.safeApprove(USDC, address(nonfungiblePositionManager), 0);
        uint256 refund1 = amount1ToMint - amount1;
        TransferHelper.safeTransfer(USDC, msg.sender, refund1);
    }
}

```

id: setting-up title: Set Up Your Contract sidebar_position: 1

Setting up the Contract

This guide is an example of a custodial contract Uniswap V3 positions, which allows interaction with the Uniswap V3 Periphery by minting a position, adding liquidity to a position, decreasing liquidity, and collecting fees.

First, declare the solidity version used to compile the contract and `abicoder v2` to allow arbitrary nested arrays and structs to be encoded and decoded in calldata, a feature we use when transacting with a pool.

```
// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity =0.7.6;
pragma abicoder v2;
```

Import the contracts needed from the npm package installation.

```
import '@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol';
import '@uniswap/v3-core/contracts/libraries/TickMath.sol';
import '@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol';
import '@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol';
import '@uniswap/v3-periphery/contracts/interfaces/INonfungiblePositionManager.sol';
import '@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol';
import '@uniswap/v3-periphery/contracts/base/LiquidityManagement.sol';
```

Create a contract called `LiquidityExamples` and inherit both `IERC721Receiver` and `LiquidityManagement`.

We've chosen to hardcode the token contract addresses and pool fee tiers for our example. In production, you would likely use an input parameter for this, allowing you to change the pools and tokens you are interacting with on a per transaction basis.

```
contract LiquidityExamples is IERC721Receiver {

    address public constant DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
    address public constant USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

    uint24 public constant poolFee = 3000;
```

Declare an immutable public variable `nonfungiblePositionManager` of type `INonfungiblePositionManager`.

```
INonfungiblePositionManager public immutable nonfungiblePositionManager;
```

Allowing ERC721 Interactions

Every [NFT](#) is identified by a unique uint256 ID inside the ERC-721 smart contract, declared as the `tokenId`

To allow deposits of ERC721 expressions of liquidity, create a struct called `Deposit`, a mapping of `uint256` to the `Deposit` struct, then declare that mapping as a public variable `deposits`.

```
struct Deposit {
    address owner;
    uint128 liquidity;
    address token0;
    address token1;
}

mapping(uint256 => Deposit) public deposits;
```

The Constructor

Declare the constructor here, which is executed once when the contract is deployed. Our constructor hard codes the address of the nonfungible position manager interface, V3 router, and the periphery immutable state constructor, which requires the factory and the address of weth9 (the [ERC-20 wrapper](#) for ether).

```
constructor(
    INonfungiblePositionManager _nonfungiblePositionManager,
    address _factory,
    address _WETH9
) PeripheryImmutableState(_factory, _WETH9) {
    nonfungiblePositionManager = _nonfungiblePositionManager;
}
```

Allowing custody of ERC721 tokens

To allow the contract to custody ERC721 tokens, implement the `onERC721Received` function within the inherited `IERC721Receiver.sol` [contract](#).

The `from` identifier may be omitted because it is not used.

```
function onERC721Received(
    address operator,
    address,
    uint256 tokenId,
    bytes calldata
) external override returns (bytes4) {
    // get position information
    _createDeposit(operator, tokenId);
    return this.onERC721Received.selector;
}
```

Creating a Deposit

To add a `Deposit` instance to the `deposits` mapping, create an internal function called `_createDeposit` that destructures the `positions` struct returned by `positions` in `nonfungiblePositionManager.sol`. Pass the relevant variables `token0`, `token1` and `liquidity` to the `deposits` mapping.

```
function _createDeposit(address owner, uint256 tokenId) internal {
    (, , address token0, address token1, , , uint128 liquidity, , , ) =
        nonfungiblePositionManager.positions(tokenId);

    // set the owner and data for position
    // operator is msg.sender
    deposits[tokenId] = Deposit({owner: owner, liquidity: liquidity, token0: token0, token1: token1});
}
```

The Full Contract Setup

```
// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity =0.7.6;
pragma abicoder v2;

import '@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol';
import '@uniswap/v3-core/contracts/libraries/TickMath.sol';
import '@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol';
import '../libraries/TransferHelper.sol';
import '../interfaces/INonfungiblePositionManager.sol';
import '../base/LiquidityManagement.sol';

contract LiquidityExamples is IERC721Receiver {
    address public constant DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
    address public constant USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

    uint24 public constant poolFee = 3000;
```

```

INonfungiblePositionManager public immutable nonfungiblePositionManager;

/// @notice Represents the deposit of an NFT
struct Deposit {
    address owner;
    uint128 liquidity;
    address token0;
    address token1;
}

/// @dev deposits[tokenId] => Deposit
mapping(uint256 => Deposit) public deposits;

constructor(
    INonfungiblePositionManager _nonfungiblePositionManager
) {
    nonfungiblePositionManager = _nonfungiblePositionManager;
}

// Implementing `onERC721Received` so this contract can receive custody of erc721 tokens
function onERC721Received(
    address operator,
    address,
    uint256 tokenId,
    bytes calldata
) external override returns (bytes4) {
    // get position information

    _createDeposit(operator, tokenId);

    return this.onERC721Received.selector;
}

function _createDeposit(address owner, uint256 tokenId) internal {
    (, , address token0, address token1, , , uint128 liquidity, , , ) =
        nonfungiblePositionManager.positions(tokenId);

    // set the owner and data for position
    // operator is msg.sender
    deposits[tokenId] = Deposit({owner: owner, liquidity: liquidity, token0: token0, token1: token1});
}
}

```

id: the-full-contract title: The Full Contract sidebar_position: 6

Below we have the complete functioning code example: a contract that can custody Uniswap V3 position NFT's and manipulate the positions and liquidity therein by collecting fees, increasing or decreasing liquidity, and minting new positions. View on github [here](#).

```

// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity =0.7.6;
pragma abicoder v2;

import '@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol';
import '@uniswap/v3-core/contracts/libraries/TickMath.sol';
import '@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol';
import '../libraries/TransferHelper.sol';
import '../interfaces/INonfungiblePositionManager.sol';
import '../base/LiquidityManagement.sol';

contract LiquidityExamples is IERC721Receiver {
    address public constant DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
    address public constant USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

    uint24 public constant poolFee = 3000;

    INonfungiblePositionManager public immutable nonfungiblePositionManager;

    /// @notice Represents the deposit of an NFT
    struct Deposit {

```

```

        address owner;
        uint128 liquidity;
        address token0;
        address token1;
    }

    /// @dev deposits[tokenId] => Deposit
    mapping(uint256 => Deposit) public deposits;

    constructor(
        INonfungiblePositionManager _nonfungiblePositionManager
    ) {
        nonfungiblePositionManager = _nonfungiblePositionManager;
    }

    // Implementing `onERC721Received` so this contract can receive custody of erc721 tokens
    function onERC721Received(
        address operator,
        address,
        uint256 tokenId,
        bytes calldata
    ) external override returns (bytes4) {
        // get position information

        _createDeposit(operator, tokenId);

        return this.onERC721Received.selector;
    }

    function _createDeposit(address owner, uint256 tokenId) internal {
        (, address token0, address token1, , , uint128 liquidity, , , ) =
            nonfungiblePositionManager.positions(tokenId);

        // set the owner and data for position
        // operator is msg.sender
        deposits[tokenId] = Deposit({owner: owner, liquidity: liquidity, token0: token0, token1: token1});
    }

    /// @notice Calls the mint function defined in periphery, mints the same amount of each token.
    /// For this example we are providing 1000 DAI and 1000 USDC in liquidity
    /// @return tokenId The id of the newly minted ERC721
    /// @return liquidity The amount of liquidity for the position
    /// @return amount0 The amount of token0
    /// @return amount1 The amount of token1
    function mintNewPosition()
        external
        returns (
            uint256 tokenId,
            uint128 liquidity,
            uint256 amount0,
            uint256 amount1
        )
    {
        // For this example, we will provide equal amounts of liquidity in both assets.
        // Providing liquidity in both assets means liquidity will be earning fees and is considered in-range.
        uint256 amount0ToMint = 1000;
        uint256 amount1ToMint = 1000;

        // transfer tokens to contract
        TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amount0ToMint);
        TransferHelper.safeTransferFrom(USDC, msg.sender, address(this), amount1ToMint);

        // Approve the position manager
        TransferHelper.safeApprove(DAI, address(nonfungiblePositionManager), amount0ToMint);
        TransferHelper.safeApprove(USDC, address(nonfungiblePositionManager), amount1ToMint);

        INonfungiblePositionManager.MintParams memory params =
            INonfungiblePositionManager.MintParams({
                token0: DAI,
                token1: USDC,

```

```

        fee: poolFee,
        tickLower: TickMath.MIN_TICK,
        tickUpper: TickMath.MAX_TICK,
        amount0Desired: amount0ToMint,
        amount1Desired: amount1ToMint,
        amount0Min: 0,
        amount1Min: 0,
        recipient: address(this),
        deadline: block.timestamp
    });

    // Note that the pool defined by DAI/USDC and fee tier 0.3% must already be created and initialized in
    order to mint
    (tokenId, liquidity, amount0, amount1) = nonfungiblePositionManager.mint(params);

    // Create a deposit
    _createDeposit(msg.sender, tokenId);

    // Remove allowance and refund in both assets.
    if (amount0 < amount0ToMint) {
        TransferHelper.safeApprove(DAI, address(nonfungiblePositionManager), 0);
        uint256 refund0 = amount0ToMint - amount0;
        TransferHelper.safeTransfer(DAI, msg.sender, refund0);
    }

    if (amount1 < amount1ToMint) {
        TransferHelper.safeApprove(USDC, address(nonfungiblePositionManager), 0);
        uint256 refund1 = amount1ToMint - amount1;
        TransferHelper.safeTransfer(USDC, msg.sender, refund1);
    }
}

/// @notice Collects the fees associated with provided liquidity
/// @dev The contract must hold the erc721 token before it can collect fees
/// @param tokenId The id of the erc721 token
/// @return amount0 The amount of fees collected in token0
/// @return amount1 The amount of fees collected in token1
function collectAllFees(uint256 tokenId) external returns (uint256 amount0, uint256 amount1) {
    // Caller must own the ERC721 position, meaning it must be a deposit

    // set amount0Max and amount1Max to uint256.max to collect all fees
    // alternatively can set recipient to msg.sender and avoid another transaction in `sendToOwner`
    INonfungiblePositionManager.CollectParams memory params =
        INonfungiblePositionManager.CollectParams({
            tokenId: tokenId,
            recipient: address(this),
            amount0Max: type(uint128).max,
            amount1Max: type(uint128).max
        });

    (amount0, amount1) = nonfungiblePositionManager.collect(params);

    // send collected feed back to owner
    _sendToOwner(tokenId, amount0, amount1);
}

/// @notice A function that decreases the current liquidity by half. An example to show how to call the
`decreaseLiquidity` function defined in periphery.
/// @param tokenId The id of the erc721 token
/// @return amount0 The amount received back in token0
/// @return amount1 The amount returned back in token1
function decreaseLiquidityInHalf(uint256 tokenId) external returns (uint256 amount0, uint256 amount1) {
    // caller must be the owner of the NFT
    require(msg.sender == deposits[tokenId].owner, 'Not the owner');
    // get liquidity data for tokenId
    uint128 liquidity = deposits[tokenId].liquidity;
    uint128 halfLiquidity = liquidity / 2;

    // amount0Min and amount1Min are price slippage checks
    // if the amount received after burning is not greater than these minimums, transaction will fail
}

```

```

INonfungiblePositionManager.DecreaseLiquidityParams memory params =
    INonfungiblePositionManager.DecreaseLiquidityParams({
        tokenId: tokenId,
        liquidity: halfLiquidity,
        amount0Min: 0,
        amount1Min: 0,
        deadline: block.timestamp
    });
}

(amount0, amount1) = nonfungiblePositionManager.decreaseLiquidity(params);

//send liquidity back to owner
_sendToOwner(tokenId, amount0, amount1);
}

/// @notice Increases liquidity in the current range
/// @dev Pool must be initialized already to add liquidity
/// @param tokenId The id of the erc721 token
/// @param amount0 The amount to add of token0
/// @param amount1 The amount to add of token1
function increaseLiquidityCurrentRange(
    uint256 tokenId,
    uint256 amountAdd0,
    uint256 amountAdd1
)
external
returns (
    uint128 liquidity,
    uint256 amount0,
    uint256 amount1
) {
    TransferHelper.safeTransferFrom(deposits[tokenId].token0, msg.sender, address(this), amountAdd0);
    TransferHelper.safeTransferFrom(deposits[tokenId].token1, msg.sender, address(this), amountAdd1);

    TransferHelper.safeApprove(deposits[tokenId].token0, address(nonfungiblePositionManager), amountAdd0);
    TransferHelper.safeApprove(deposits[tokenId].token1, address(nonfungiblePositionManager), amountAdd1);

    INonfungiblePositionManager.IncreaseLiquidityParams memory params =
    INonfungiblePositionManager.IncreaseLiquidityParams({
        tokenId: tokenId,
        amount0Desired: amountAdd0,
        amount1Desired: amountAdd1,
        amount0Min: 0,
        amount1Min: 0,
        deadline: block.timestamp
    });

    (liquidity, amount0, amount1) = nonfungiblePositionManager.increaseLiquidity(params);
}

/// @notice Transfers funds to owner of NFT
/// @param tokenId The id of the erc721
/// @param amount0 The amount of token0
/// @param amount1 The amount of token1
function _sendToOwner(
    uint256 tokenId,
    uint256 amount0,
    uint256 amount1
) internal {
    // get owner of contract
    address owner = deposits[tokenId].owner;

    address token0 = deposits[tokenId].token0;
    address token1 = deposits[tokenId].token1;
    // send collected fees to owner
    TransferHelper.safeTransfer(token0, owner, amount0);
    TransferHelper.safeTransfer(token1, owner, amount1);
}

```

```

    /// @notice Transfers the NFT to the owner
    /// @param tokenId The id of the erc721
    function retrieveNFT(uint256 tokenId) external {
        // must be the owner of the NFT
        require(msg.sender == deposits[tokenId].owner, 'Not the owner');
        // transfer ownership to original owner
        nonfungiblePositionManager.safeTransferFrom(address(this), msg.sender, tokenId);
        //remove information related to tokenId
        delete deposits[tokenId];
    }
}

```

id: multihop-swaps title: Multihop Swaps sidebar_position: 2

Introduction

The examples below are implementations of the two styles of multi-hop swapping available on v3. The examples below are not production ready code, and are implemented in a simplistic manner for the purpose of learning.

Setting up the Contract

Declare the solidity version that will be used to compile the contract, and the `abicoder v2` to allow arbitrary nested arrays and structs to be encoded and decoded in calldata, a feature we use when executing a swap.

```

// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity =0.7.6;
pragma abicoder v2;

```

Import the two needed contracts from the npm package installation.

```

import '@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol';
import '@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol';

```

Create a contract called `SwapExamples`, and declare an immutable public variable `swapRouter` of type `ISwapRouter`. This allows us to call functions in the `ISwapRouter` interface.

```

contract SwapExamples {
    // For the scope of these swap examples,
    // we will detail the design considerations when using `exactInput`, `exactInputSingle`, `exactOutput`, and
    `exactOutputSingle`.
    // It should be noted that for the sake of these examples we pass in the swap router as a constructor argument
    instead of inheriting it.
    // More advanced example contracts will detail how to inherit the swap router safely.
    // This example swaps DAI/WETH9 for single path swaps and DAI/USDC/WETH9 for multi path swaps.

    ISwapRouter public immutable swapRouter;
}

```

Hardcode the token contract addresses and pool fee tiers for the example. In production, you would likely use an input parameter for this and pass the input into a memory variable, allowing the contract to change the pools and tokens it interacts with on a per transaction basis, but for conceptual simplicity, we are hardcoding them here.

```

address public constant DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
address public constant WETH9 = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
address public constant USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

// For this example, we will set the pool fee to 0.3%.
uint24 public constant poolFee = 3000;

constructor(ISwapRouter _swapRouter) {
    swapRouter = _swapRouter;
}

```

Exact Input Multi Hop Swaps

Exact input multi hop swaps will swap a fixed amount on a given input token for the maximum amount possible for a given output, and can include an arbitrary number of intermediary swaps.

Input Parameters

- `path` : The path is a sequence of (`tokenAddress - fee - tokenAddress`), which are the variables needed to compute each pool contract address in our sequence of swaps. The multihop swap router code will automatically find the correct pool with these variables, and execute the swap needed within each pool in our sequence.
- `recipient` : the destination address of the outbound asset.
- `deadline` : the unix time after which a transaction will be reverted, to protect against long delays and the increased chance of large price swings therein.
- `amountIn` : the amount of the inbound asset
- `amountOutMin` : the minimum amount of the outbound asset, less than which will cause the transaction to revert. For the sake of this example we will set it to `0`, in production one will need to use the SDK to quote an expected price, or an on chain price oracle for more advanced manipulation resistant systems.

Calling the function

```
/// @notice swapExactInputMultihop swaps a fixed amount of DAI for a maximum possible amount of WETH9 through
an intermediary pool.
/// For this example, we will swap DAI to USDC, then USDC to WETH9 to achieve our desired output.
/// @dev The calling address must approve this contract to spend at least `amountIn` worth of its DAI for this
function to succeed.
/// @param amountIn The amount of DAI to be swapped.
/// @return amountOut The amount of WETH9 received after the swap.
function swapExactInputMultihop(uint256 amountIn) external returns (uint256 amountOut) {
    // Transfer `amountIn` of DAI to this contract.
    TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amountIn);

    // Approve the router to spend DAI.
    TransferHelper.safeApprove(DAI, address(swapRouter), amountIn);

    // Multiple pool swaps are encoded through bytes called a `path`. A path is a sequence of token addresses
    and poolFees that define the pools used in the swaps.
    // The format for pool encoding is (tokenIn, fee, tokenOut/tokenIn, fee, tokenOut) where tokenIn/tokenOut
    parameter is the shared token across the pools.
    // Since we are swapping DAI to USDC and then USDC to WETH9 the path encoding is (DAI, 0.3%, USDC, 0.3%,
    WETH9).
    ISwapRouter.ExactInputParams memory params =
        ISwapRouter.ExactInputParams({
            path: abi.encodePacked(DAI, poolFee, USDC, poolFee, WETH9),
            recipient: msg.sender,
            deadline: block.timestamp,
            amountIn: amountIn,
            amountOutMinimum: 0
        });

    // Executes the swap.
    amountOut = swapRouter.exactInput(params);
}
```

Exact Output Multihop Swap

An exact output swap will swap a variable amount of the input token for a fixed amount of the outbound token. This is the less common technique for multihop swaps. The code for swapping is largely the same except for one notable difference, the `Path` is encoded backwards, as an exact output swap is executed in reverse order to pass down the necessary variables for the chain of transactions

Input Parameters

- `path` : The path is a sequence of `tokenAddress | Fee | tokenAddress`, *encoded in reverse order*, which are the variables needed to compute each pool contract address in our sequence of swaps. The multihop swap router code will automatically find the correct pool with these variables, and execute the swap needed within each pool in our sequence.
- `recipient` : the destination address of the outbound asset.
- `deadline` : the unix time after which a transaction will be reverted, to protect against long delays and the increased chance of large price swings therein.
- `amountOut` : The desired amount of WETH9.
- `amountInMaximum` : The maximum amount of DAI willing to be swapped for the specified `amountOut` of WETH9.

Calling the function

```

    /// @notice swapExactOutputMultiHop swaps a minimum possible amount of DAI for a fixed amount of WETH through
    an intermediary pool.
    /// For this example, we want to swap DAI for WETH9 through a USDC pool but we specify the desired amountOut of
    WETH9. Notice how the path encoding is slightly different in for exact output swaps.
    /// @dev The calling address must approve this contract to spend its DAI for this function to succeed. As the
    amount of input DAI is variable,
    /// the calling address will need to approve for a slightly higher amount, anticipating some variance.
    /// @param amountOut The desired amount of WETH9.
    /// @param amountInMaximum The maximum amount of DAI willing to be swapped for the specified amountOut of
    WETH9.
    /// @return amountIn The amountIn of DAI actually spent to receive the desired amountOut.
    function swapExactOutputMultiHop(uint256 amountOut, uint256 amountInMaximum) external returns (uint256
amountIn) {
    // Transfer the specified `amountInMaximum` to this contract.
    TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amountInMaximum);
    // Approve the router to spend `amountInMaximum`.
    TransferHelper.safeApprove(DAI, address(swapRouter), amountInMaximum);

    // The parameter path is encoded as (tokenOut, fee, tokenIn/tokenOut, fee, tokenIn)
    // The tokenIn/tokenOut field is the shared token between the two pools used in the multiple pool swap. In
    this case USDC is the "shared" token.
    // For an exactOutput swap, the first swap that occurs is the swap which returns the eventual desired
    token.
    // In this case, our desired output token is WETH9 so that swap happens first, and is encoded in the path
    accordingly.
    ISwapRouter.ExactOutputParams memory params =
    ISwapRouter.ExactOutputParams({
        path: abi.encodePacked(WETH9, poolFee, USDC, poolFee, DAI),
        recipient: msg.sender,
        deadline: block.timestamp,
        amountOut: amountOut,
        amountInMaximum: amountInMaximum
    });
    // Executes the swap, returning the amountIn actually spent.
    amountIn = swapRouter.exactOutput(params);

    // If the swap did not require the full amountInMaximum to achieve the exact amountOut then we refund
    msg.sender and approve the router to spend 0.
    if (amountIn < amountInMaximum) {
        TransferHelper.safeApprove(DAI, address(swapRouter), 0);
        TransferHelper.safeTransferFrom(DAI, address(this), msg.sender, amountInMaximum - amountIn);
    }
}

```

The Full Contract

```

// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity =0.7.6;
pragma abicoder v2;

import '@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol';
import '@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol';

contract SwapExamples {
    // For the scope of these swap examples,
    // we will detail the design considerations when using
    // `exactInput`, `exactInputSingle`, `exactOutput`, and `exactOutputSingle`.

    // It should be noted that for the sake of these examples, we purposefully pass in the swap router instead of
    inherit the swap router for simplicity.
    // More advanced example contracts will detail how to inherit the swap router safely.

    ISwapRouter public immutable swapRouter;

    // This example swaps DAI/WETH9 for single path swaps and DAI/USDC/WETH9 for multi path swaps.

    address public constant DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;

```

```

address public constant WETH9 = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
address public constant USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

// For this example, we will set the pool fee to 0.3%.
uint24 public constant poolFee = 3000;

constructor(ISwapRouter _swapRouter) {
    swapRouter = _swapRouter;
}

/// @notice swapInputMultiplePools swaps a fixed amount of DAI for a maximum possible amount of WETH9 through
an intermediary pool.
/// For this example, we will swap DAI to USDC, then USDC to WETH9 to achieve our desired output.
/// @dev The calling address must approve this contract to spend at least `amountIn` worth of its DAI for this
function to succeed.
/// @param amountIn The amount of DAI to be swapped.
/// @return amountOut The amount of WETH9 received after the swap.
function swapExactInputMultihop(uint256 amountIn) external returns (uint256 amountOut) {
    // Transfer `amountIn` of DAI to this contract.
    TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amountIn);

    // Approve the router to spend DAI.
    TransferHelper.safeApprove(DAI, address(swapRouter), amountIn);

    // Multiple pool swaps are encoded through bytes called a `path`. A path is a sequence of token addresses
    and poolFees that define the pools used in the swaps.
    // The format for pool encoding is (tokenIn, fee, tokenOut/tokenIn, fee, tokenOut) where tokenIn/tokenOut
    parameter is the shared token across the pools.
    // Since we are swapping DAI to USDC and then USDC to WETH9 the path encoding is (DAI, 0.3%, USDC, 0.3%,
    WETH9).
    ISwapRouter.ExactInputParams memory params =
        ISwapRouter.ExactInputParams({
            path: abi.encodePacked(DAI, poolFee, USDC, poolFee, WETH9),
            recipient: msg.sender,
            deadline: block.timestamp,
            amountIn: amountIn,
            amountOutMinimum: 0
        });

    // Executes the swap.
    amountOut = swapRouter.exactInput(params);
}

/// @notice swapExactOutputMultihop swaps a minimum possible amount of DAI for a fixed amount of WETH through
an intermediary pool.
/// For this example, we want to swap DAI for WETH9 through a USDC pool but we specify the desired amountOut of
WETH9. Notice how the path encoding is slightly different in for exact output swaps.
/// @dev The calling address must approve this contract to spend its DAI for this function to succeed. As the
amount of input DAI is variable,
/// the calling address will need to approve for a slightly higher amount, anticipating some variance.
/// @param amountOut The desired amount of WETH9.
/// @param amountInMaximum The maximum amount of DAI willing to be swapped for the specified amountOut of
WETH9.
/// @return amountIn The amountIn of DAI actually spent to receive the desired amountOut.
function swapExactOutputMultihop(uint256 amountOut, uint256 amountInMaximum) external returns (uint256
amountIn) {
    // Transfer the specified `amountInMaximum` to this contract.
    TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amountInMaximum);
    // Approve the router to spend `amountInMaximum`.
    TransferHelper.safeApprove(DAI, address(swapRouter), amountInMaximum);

    // The parameter path is encoded as (tokenOut, fee, tokenIn/tokenOut, fee, tokenIn)
    // The tokenIn/tokenOut field is the shared token between the two pools used in the multiple pool swap. In
    this case USDC is the "shared" token.
    // For an exactOutput swap, the first swap that occurs is the swap which returns the eventual desired
    token.
    // In this case, our desired output token is WETH9 so that swap happens first, and is encoded in the path
    accordingly.
    ISwapRouter.ExactOutputParams memory params =
        ISwapRouter.ExactOutputParams({

```

```

        path: abi.encodePacked(WETH9, poolFee, USDC, poolFee, DAI),
        recipient: msg.sender,
        deadline: block.timestamp,
        amountOut: amountOut,
        amountInMaximum: amountInMaximum
    });

    // Executes the swap, returning the amountIn actually spent.
    amountIn = swapRouter.exactOutput(params);

    // If the swap did not require the full amountInMaximum to achieve the exact amountOut then we refund
    msg.sender and approve the router to spend 0.
    if (amountIn < amountInMaximum) {
        TransferHelper.safeApprove(DAI, address(swapRouter), 0);
        TransferHelper.safeTransferFrom(DAI, address(this), msg.sender, amountInMaximum - amountIn);
    }
}
}
}

```

id: single-swaps title: Single Swaps sidebar_position: 1

Swaps are the most common interaction with the Uniswap protocol. The following example shows you how to implement a single-path swap contract that uses two functions that you create:

- `swapExactInputSingle`
- `swapExactOutputSingle`

The `swapExactInputSingle` function is for performing *exact input* swaps, which swap a fixed amount of one token for a maximum possible amount of another token. This function uses the `ExactInputSingleParams` struct and the `exactInputSingle` function from the [ISwapRouter](#) interface.

The `swapExactOutputSingle` function is for performing *exact output* swaps, which swap a minimum possible amount of one token for a fixed amount of another token. This function uses the `ExactOutputSingleParams` struct and the `exactOutputSingle` function from the [ISwapRouter](#) interface.

For simplification, the example hardcodes the token contract addresses, but as explained further below the contract could be modified to change pools and tokens on a per transaction basis.

When trading from a smart contract, the most important thing to keep in mind is that access to an external price source is required. Without this, trades can be frontrun for considerable loss.

Note: The swap examples are not production ready code, and are implemented in a simplistic manner for the purpose of learning.

Set Up the Contract

Declare the solidity version used to compile the contract, and `abicoder v2` to allow arbitrary nested arrays and structs to be encoded and decoded in calldata, a feature used when executing a swap.

```
// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity =0.7.6;
pragma abicoder v2;
```

Import the two relevant contracts from the npm package installation

```
import '@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol';
import '@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol';
```

Create a contract called `SwapExamples`, and declare an immutable public variable `swapRouter` of type `ISwapRouter`. This allows us to call functions in the `ISwapRouter` interface.

```
contract SwapExamples {
    // For the scope of these swap examples,
    // we will detail the design considerations when using `exactInput`, `exactInputSingle`, `exactOutput`, and
    `exactOutputSingle`.
    // It should be noted that for the sake of these examples we pass in the swap router as a constructor argument
    instead of inheriting it.
    // More advanced example contracts will detail how to inherit the swap router safely.
    // This example swaps DAI/WETH9 for single path swaps and DAI/USDC/WETH9 for multi path swaps.
```

```
ISwapRouter public immutable swapRouter;
```

Hardcode the token contract addresses and pool fee tiers for the example. In production, you would likely use an input parameter for this and pass the input into a memory variable, allowing the contract to change the pools and tokens it interacts with on a per transaction basis, but for conceptual simplicity, we are hardcoding them here.

```
address public constant DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
address public constant WETH9 = 0xC02aaA39b223F8D0A0e5C4F27eAD9083C756Cc2;
address public constant USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

// For this example, we will set the pool fee to 0.3%.
uint24 public constant poolFee = 3000;

constructor(ISwapRouter _swapRouter) {
    swapRouter = _swapRouter;
}
```

Exact Input Swaps

The caller must `approve` the contract to withdraw the tokens from the calling address's account to execute a swap. Remember that because our contract is a contract itself and not an extension of the caller (us); we must also approve the Uniswap protocol router contract to use the tokens that our contract will be in possession of after they have been withdrawn from the calling address (us).

Then, transfer the `amount` of Dai from the calling address into our contract, and use `amount` as the value passed to the second `approve`.

```
/// @notice swapExactInputSingle swaps a fixed amount of DAI for a maximum possible amount of WETH9
/// using the DAI/WETH9 0.3% pool by calling `exactInputSingle` in the swap router.
/// @dev The calling address must approve this contract to spend at least `amountIn` worth of its DAI for this
function swapExactInputSingle(uint256 amountIn) external returns (uint256 amountOut) {
    // msg.sender must approve this contract

    // Transfer the specified amount of DAI to this contract.
    TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amountIn);

    // Approve the router to spend DAI.
    TransferHelper.safeApprove(DAI, address(swapRouter), amountIn);
```

Swap Input Parameters

To execute the swap function, we need to populate the `ExactInputSingleParams` with the necessary swap data. These parameters are found in the smart contract interfaces, which can be browsed [here](#).

A brief overview of the parameters:

- `tokenIn` The contract address of the inbound token
- `tokenOut` The contract address of the outbound token
- `fee` The fee tier of the pool, used to determine the correct pool contract in which to execute the swap
- `recipient` the destination address of the outbound token
- `deadline` : the unix time after which a swap will fail, to protect against long-pending transactions and wild swings in prices
- `amountOutMinimum` : we are setting to zero, but this is a significant risk in production. For a real deployment, this value should be calculated using our SDK or an onchain price oracle - this helps protect against getting an unusually bad price for a trade due to a front running sandwich or another type of price manipulation
- `sqrtPriceLimitX96` : We set this to zero - which makes this parameter inactive. In production, this value can be used to set the limit for the price the swap will push the pool to, which can help protect against price impact or for setting up logic in a variety of price-relevant mechanisms.

Call the function

```
// Naively set amountOutMinimum to 0. In production, use an oracle or other data source to choose a safer
value for amountOutMinimum.
// We also set the sqrtPriceLimitx96 to be 0 to ensure we swap our exact input amount.
ISwapRouter.ExactInputSingleParams memory params =
    ISwapRouter.ExactInputSingleParams({
        tokenIn: DAI,
```

```

        tokenOut: WETH9,
        fee: poolFee,
        recipient: msg.sender,
        deadline: block.timestamp,
        amountIn: amountIn,
        amountOutMinimum: 0,
        sqrtPriceLimitX96: 0
    });

    // The call to `exactInputSingle` executes the swap.
    amountOut = swapRouter.exactInputSingle(params);
}

```

Exact Output Swaps

Exact Output swaps a minimum possible amount of the input token for a fixed amount of the outbound token. This is the less common swap style - but useful in a variety of circumstances.

Because this example transfers in the inbound asset in anticipation of the swap - its possible that some of the inbound token will be left over after the swap is executed, which is why we pay it back to the calling address at the end of the swap.

Call the function

```

/// @notice swapExactOutputSingle swaps a minimum possible amount of DAI for a fixed amount of WETH.
/// @dev The calling address must approve this contract to spend its DAI for this function to succeed. As the
amount of input DAI is variable,
/// the calling address will need to approve for a slightly higher amount, anticipating some variance.
/// @param amountOut The exact amount of WETH9 to receive from the swap.
/// @param amountInMaximum The amount of DAI we are willing to spend to receive the specified amount of WETH9.
/// @return amountIn The amount of DAI actually spent in the swap.
function swapExactOutputSingle(uint256 amountOut, uint256 amountInMaximum) external returns (uint256 amountIn) {
    // Transfer the specified amount of DAI to this contract.
    TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amountInMaximum);

    // Approve the router to spend the specified `amountInMaximum` of DAI.
    // In production, you should choose the maximum amount to spend based on oracles or other data sources to
    achieve a better swap.
    TransferHelper.safeApprove(DAI, address(swapRouter), amountInMaximum);

    ISwapRouter.ExactOutputSingleParams memory params =
        ISwapRouter.ExactOutputSingleParams({
            tokenIn: DAI,
            tokenOut: WETH9,
            fee: poolFee,
            recipient: msg.sender,
            deadline: block.timestamp,
            amountOut: amountOut,
            amountInMaximum: amountInMaximum,
            sqrtPriceLimitX96: 0
        });

    // Executes the swap returning the amountIn needed to spend to receive the desired amountOut.
    amountIn = swapRouter.exactOutputSingle(params);

    // For exact output swaps, the amountInMaximum may not have all been spent.
    // If the actual amount spent (amountIn) is less than the specified maximum amount, we must refund the
msg.sender and approve the swapRouter to spend 0.
    if (amountIn < amountInMaximum) {
        TransferHelper.safeApprove(DAI, address(swapRouter), 0);
        TransferHelper.safeTransfer(DAI, msg.sender, amountInMaximum - amountIn);
    }
}

```

A Complete Single Swap Contract

```

// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity =0.7.6;
pragma abicoder v2;

```

```

import '@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol';
import '@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol';

contract SwapExamples {
    // For the scope of these swap examples,
    // we will detail the design considerations when using
    // `exactInput`, `exactInputSingle`, `exactOutput`, and `exactOutputSingle`.

    // It should be noted that for the sake of these examples, we purposefully pass in the swap router instead of
    // inherit the swap router for simplicity.
    // More advanced example contracts will detail how to inherit the swap router safely.

    ISwapRouter public immutable swapRouter;

    // This example swaps DAI/WETH9 for single path swaps and DAI/USDC/WETH9 for multi path swaps.

    address public constant DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
    address public constant WETH9 = 0xC02aa39b223FE8D0A0e5C4F27eAD9083C756Cc2;
    address public constant USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

    // For this example, we will set the pool fee to 0.3%.
    uint24 public constant poolFee = 3000;

    constructor(ISwapRouter _swapRouter) {
        swapRouter = _swapRouter;
    }

    /// @notice swapExactInputSingle swaps a fixed amount of DAI for a maximum possible amount of WETH9
    /// using the DAI/WETH9 0.3% pool by calling `exactInputSingle` in the swap router.
    /// @dev The calling address must approve this contract to spend at least `amountIn` worth of its DAI for this
    function swapExactInputSingle(uint256 amountIn) external returns (uint256 amountOut) {
        // msg.sender must approve this contract

        // Transfer the specified amount of DAI to this contract.
        TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amountIn);

        // Approve the router to spend DAI.
        TransferHelper.safeApprove(DAI, address(swapRouter), amountIn);

        // Naively set amountOutMinimum to 0. In production, use an oracle or other data source to choose a safer
        value for amountOutMinimum.
        // We also set the sqrtPriceLimitX96 to be 0 to ensure we swap our exact input amount.
        ISwapRouter.ExactInputSingleParams memory params =
            ISwapRouter.ExactInputSingleParams({
                tokenIn: DAI,
                tokenOut: WETH9,
                fee: poolFee,
                recipient: msg.sender,
                deadline: block.timestamp,
                amountIn: amountIn,
                amountOutMinimum: 0,
                sqrtPriceLimitX96: 0
            });

        // The call to `exactInputSingle` executes the swap.
        amountOut = swapRouter.exactInputSingle(params);
    }

    /// @notice swapExactOutputSingle swaps a minimum possible amount of DAI for a fixed amount of WETH.
    /// @dev The calling address must approve this contract to spend its DAI for this function to succeed. As the
    amount of input DAI is variable,
    // the calling address will need to approve for a slightly higher amount, anticipating some variance.
    // @param amountOut The exact amount of WETH9 to receive from the swap.
    // @param amountInMaximum The amount of DAI we are willing to spend to receive the specified amount of WETH9.
    // @return amountIn The amount of DAI actually spent in the swap.
    function swapExactOutputSingle(uint256 amountOut, uint256 amountInMaximum) external returns (uint256 amountIn)
}

```

```

{
    // Transfer the specified amount of DAI to this contract.
    TransferHelper.safeTransferFrom(DAI, msg.sender, address(this), amountInMaximum);

    // Approve the router to spend the specified `amountInMaximum` of DAI.
    // In production, you should choose the maximum amount to spend based on oracles or other data sources to
    // achieve a better swap.
    TransferHelper.safeApprove(DAI, address(swapRouter), amountInMaximum);

    ISwapRouter.ExactOutputSingleParams memory params =
        ISwapRouter.ExactOutputSingleParams({
            tokenIn: DAI,
            tokenOut: WETH9,
            fee: poolFee,
            recipient: msg.sender,
            deadline: block.timestamp,
            amountOut: amountOut,
            amountInMaximum: amountInMaximum,
            sqrtPriceLimitX96: 0
        });

    // Executes the swap returning the amountIn needed to spend to receive the desired amountOut.
    amountIn = swapRouter.exactOutputSingle(params);

    // For exact output swaps, the amountInMaximum may not have all been spent.
    // If the actual amount spent (amountIn) is less than the specified maximum amount, we must refund the
    msg.sender and approve the swapRouter to spend 0.
    if (amountIn < amountInMaximum) {
        TransferHelper.safeApprove(DAI, address(swapRouter), 0);
        TransferHelper.safeTransfer(DAI, msg.sender, amountInMaximum - amountIn);
    }
}
}
}

```

id: overview title: Overview sidebar_position: 1

The Uniswap V3 Smart Contracts

Welcome to the Uniswap V3 smart contracts documentation.

The pages here contain guides and technical documentation for the Uniswap V3 Smart Contracts. You can use these docs to learn about the V3 Protocol Smart Contracts and develop on-chain integrations.

Guides

If you are new to the Uniswap Protocol, we recommend you start with the [basic concepts](#) first.

You can then setup your [local environment](#) and execute your [first swap](#).

Reference

For a deeper dive, read through the [technical reference](#) docs.

Resources

- [V3 Core](#)
- [V3 Periphery](#)-- id: deployments title: Deployment Addresses

Uniswap Contract Deployments

The latest version of `@uniswap/v3-core`, `@uniswap/v3-periphery`, and `@uniswap/swap-router-contracts` are deployed at the addresses listed below. Integrators should **no longer assume that they are deployed to the same addresses across chains** and be extremely careful to confirm mappings below.

| Contract | Mainnet, Goerli, Arbitrum, Optimism, Polygon Address | Celo Address | |
|--|--|---|-----------|
| UniswapV3Factory | 0x1F98431c8aD98523631AE4a59f267346ea31F984 | 0xAFEB208a311B21f13EF87E33A90049fc17A7acDEc | 0xdB1d100 |
| Multicall2 | 0x5BA1e12693Dc8F9c48aAD8770482f4739bEeD696 | 0x633987602DE5C4F337e3Dbf265303A1080324204 | 0x963df24 |
| ProxyAdmin | 0xB753548F6E010e7e680BA186F9Ca1BdAB2E90cf2 | 0xc1b262Dd7643D4B7cA9e51631bBd900a564BF49A | 0xC9A7f5b |
| TickLens | 0xbfd8137f7d1516D3ea5cA83523914859ec47F573 | 0x5f115D9113F88e0a0Db1b5033D90D4a9690AcD3D | 0xD927001 |
| Quoter | 0xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6 | 0x82825d0554fa07f7FC52Ab63c961F330fdEFa8E8 | |
| SwapRouter | 0xE592427A0AEce92De3Edee1F18E0157C05861564 | 0x5615CDAb10dc425a742d643d949a7F474C01abc4 | |
| NFTDescriptor | 0x42B24A95702b9986e82d421cC3568932790A48Ec | 0xa9Fd765d85938D278cb0b108DbE4BF7186831186 | 0x831d93E |
| NonfungibleTokenPositionDescriptor | 0x91ae842A5FFd8d12023116943e72A606179294f3 | 0x644023b316B65175C347DE903B60a756F6dd554 | 0x0281E98 |
| TransparentUpgradeableProxy | 0xEe6A57eC80ea46401049E92587E52f5Ec1c24785 | 0x505B43c452AA4443e0a6B84bb37771494633Fde9 | 0xAec98e4 |
| NonfungiblePositionManager | 0xC36442b4a4522E871399CD717aBDD847Ab11FE88 | 0x3d79EdAaBC0EaB6F08ED885C05Fc0B014290D95A | 0x7b8A01E |
| V3Migrator | 0xA5644E29708357803b5A882D272c41cC0dF92B34 | 0x3cPd4d48EdfDCC53D3f173F596f621064614C582 | 0x3268181 |
| QuoterV2 | 0x61fF014bA17989E743c5F6cB21bF9697530B21e | 0x82825d0554fa07f7FC52Ab63c961F330fdEFa8E8 | 0x78D78E4 |
| SwapRouter02 | 0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45 | 0x5615CDAb10dc425a742d643d949a7F474C01abc4 | 0xB971eF8 |
| Permit2 | 0x000000000022d473030f116ddee9f6b43ac78ba3 | 0x000000000022d473030f116ddee9f6b43ac78ba3 | 0x0000000 |

These addresses are final and were deployed from these npm package versions:

- [@uniswap/v3-core@1.0.0](#)
- [@uniswap/v3-periphery@1.0.0](#)
- [@uniswap/swap-router-contracts@1.1.0](#)

Uniswap V3 Staker

An up-to-date list of [deploy addresses by chain is hosted on Github](#) for the `UniswapV3Staker` contract.

Universal Router

The `UniversalRouter` contract is the current preferred entrypoint for ERC20 and NFT swaps, replacing, among other contracts, `SwapRouter02`. An up-to-date list of [deploy addresses by chain is hosted on Github](#).

Uniswap Pool Deployments

Every Uniswap pool is a unique instance of the `UniswapV3Pool` contract and is deployed at its own unique address. The contract source code of the pool will be auto-verified on etherscan. For example, here is the [ETH/USDC 0.3% pool](#) on Ethereum mainnet.

You can look up the address of an existing pool on [Uniswap Info](#) or by calling the `getPool` function on the `UniswapV3Factory` contract.

```
getPool("0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48", "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2", 3000)
```

Wrapped Native Token Addresses

The Uniswap Protocol supports trading of ERC20 tokens. In order to swap a native asset like ETH (or MATIC on Polygon), the Uniswap protocol wraps these assets in an ERC20 wrapped native token contract. The protocol uses the following WETH9 addresses on Ethereum and WMATIC addresses on Polygon.

| Network | ChainId | Wrapped Native Token | Address |
|----------|---------|----------------------|--|
| Ethereum | 1 | WETH | 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 |
| Goerli | 5 | WETH | 0xB4FBF271143F4FBf7B91A5ded31805e42b2208d6 |
| Arbitrum | 42161 | WETH | 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1 |

| | | | |
|--|--------|--------|--|
| Arbitrum Goerli | 421613 | WETH | 0xe39Ab88f8A4777030A534146A9Ca3B52bd5D43A3 |
| Optimism | 10 | WETH | 0x4200 |
| Optimism Goerli | 420 | WETH | 0x4200 |
| Polygon | 137 | WMATIC | 0x0d500B1d8E8eF31E21C99d1Db9A6444d3ADf1270 |
| Polygon Mumbai | 80001 | WMATIC | 0x9c3C9283D3e44854697Cd22D3Faa240Cfb032889 |
| Binance Smart Chain | 56 | WBONB | 0xbb4Cd9CBd36B01bD1cBaEBF2De08d9173bc095c |
| Deploys Uniswap V3 pools and manages ownership and control over pool protocol fees | | | |

Functions

createPool

```
function createPool(
    address tokenA,
    address tokenB,
    uint24 fee
) external returns (address pool)
```

Creates a pool for the given two tokens and fee

tokenA and tokenB may be passed in either order: token0/token1 or token1/token0. tickSpacing is retrieved from the fee. The call will revert if the pool already exists, the fee is invalid, or the token arguments are invalid.

Parameters:

| Name | Type | Description |
|--------|---------|---|
| tokenA | address | One of the two tokens in the desired pool |
| tokenB | address | The other of the two tokens in the desired pool |
| fee | uint24 | The desired fee for the pool |

Return Values:

| Name | Type | Description |
|------|---------|---------------------------------------|
| pool | address | The address of the newly created pool |

setOwner

```
function setOwner(
    address _owner
) external
```

Updates the owner of the factory

Must be called by the current owner

Parameters:

| Name | Type | Description |
|--------|---------|------------------------------|
| _owner | address | The new owner of the factory |

enableFeeAmount

```
function enableFeeAmount(
    uint24 fee,
    int24 tickSpacing
) public
```

Enables a fee amount with the given tickSpacing

Fee amounts may never be removed once enabled

Parameters:

| Name | Type | Description |
|-------------|--------|--|
| fee | uint24 | The fee amount to enable, denominated in hundredths of a bip (i.e. 1e-6) |
| tickSpacing | int24 | The spacing between ticks to be enforced for all pools created with the given fee amount |

Functions

_blockTimestamp

```
function _blockTimestamp(
) internal view virtual returns (uint32)
```

Returns the block timestamp truncated to 32 bits, i.e. mod 2^{32} . This method is overridden in tests.

snapshotCumulativesInside

```
function snapshotCumulativesInside(
    int24 tickLower,
    int24 tickUpper
) external view override noDelegateCall returns (int56 tickCumulativeInside, uint160
secondsPerLiquidityInsideX128, uint32 secondsInside)
```

Returns a snapshot of the tick cumulative, seconds per liquidity and seconds inside a tick range

Snapshots must only be compared to other snapshots, taken over a period for which a position existed. I.e., snapshots cannot be compared if a position is not held for the entire period between when the first snapshot is taken and the second snapshot is taken.

Parameters:

| Name | Type | Description |
|-----------|-------|-----------------------------|
| tickLower | int24 | The lower tick of the range |
| tickUpper | int24 | The upper tick of the range |

Return Values:

| Name | Type | Description |
|-------------------------------|---------|---|
| tickCumulativeInside | int56 | The snapshot of the tick accumulator for the range |
| secondsPerLiquidityInsideX128 | uint160 | The snapshot of seconds per liquidity for the range |
| secondsInside | uint32 | The snapshot of seconds per liquidity for the range |

observe

```
function observe(
    uint32[] secondsAgo
) external view override noDelegateCall returns (int56[] tickCumulatives, uint160[]
secondsPerLiquidityCumulativeX128s)
```

Returns the cumulative tick and liquidity as of each timestamp `secondsAgo` from the current block timestamp

To get a time weighted average tick or liquidity-in-range, you must call this with two values, one representing the beginning of the period and another for the end of the period. E.g., to get the last hour time-weighted average tick, you must call it with `secondsAgo = [3600, 0]`. The time weighted average tick represents the geometric time weighted average price of the pool, in log base $\sqrt{1.0001}$ of `token1 / token0`. The `TickMath` library can be used to go from a tick value to a ratio.

Parameters:

| Name | Type | Description |
|------------|----------|---|
| secondsAgo | uint32[] | From how long ago each cumulative tick and liquidity value should be returned |

Return Values:

| Name | Type | Description |
|------------------------------------|-----------|---|
| tickCumulatives | int56[] | Cumulative tick values as of each <code>secondsAgo</code> from the current block timestamp |
| secondsPerLiquidityCumulativeX128s | uint160[] | Cumulative seconds per liquidity-in-range value as of each <code>secondsAgo</code> from the current block |

timestamp

increaseObservationCardinalityNext

```
function increaseObservationCardinalityNext(
    uint16 observationCardinalityNext
) external override lock noDelegateCall
```

Increase the maximum number of price and liquidity observations that this pool will store

This method is no-op if the pool already has an `observationCardinalityNext` greater than or equal to the input `observationCardinalityNext`.**Parameters:**

| Name | Type | Description |
|---|--------|--|
| <code>observationCardinalityNext</code> | uint16 | The desired minimum number of observations for the pool to store |

initialize

```
function initialize(
    uint160 sqrtPriceX96
) external override
```

Sets the initial price for the pool

not locked because it initializes unlocked

Parameters:

| Name | Type | Description |
|---------------------------|---------|--|
| <code>sqrtPriceX96</code> | uint160 | the initial sqrt price of the pool as a Q64.96 |

mint

```
function mint(
    address recipient,
    int24 tickLower,
    int24 tickUpper,
    uint128 amount,
    bytes data
) external override lock returns (uint256 amount0, uint256 amount1)
```

Adds liquidity for the given recipient/tickLower/tickUpper position

`noDelegateCall` is applied indirectly via `_modifyPosition`**Parameters:**

| Name | Type | Description |
|------------------------|---------|--|
| <code>recipient</code> | address | The address for which the liquidity will be created |
| <code>tickLower</code> | int24 | The lower tick of the position in which to add liquidity |
| <code>tickUpper</code> | int24 | The upper tick of the position in which to add liquidity |
| <code>amount</code> | uint128 | The amount of liquidity to mint |
| <code>data</code> | bytes | Any data that should be passed through to the callback |

Return Values:

| Name | Type | Description |
|---------|---------|---|
| amount0 | uint256 | The amount of token0 that was paid to mint the given amount of liquidity. Matches the value in the callback |
| amount1 | uint256 | The amount of token1 that was paid to mint the given amount of liquidity. Matches the value in the callback |

collect

```
function collect(
    address recipient,
    int24 tickLower,
    int24 tickUpper,
    uint128 amount0Requested,
    uint128 amount1Requested
) external override lock returns (uint128 amount0, uint128 amount1)
```

Collects tokens owed to a position

Does not recompute fees earned, which must be done either via mint or burn of any amount of liquidity. Collect must be called by the position owner. To withdraw only token0 or only token1, amount0Requested or amount1Requested may be set to zero. To withdraw all tokens owed, caller may pass any value greater than the actual tokens owed, e.g. type(uint128).max. Tokens owed may be from accumulated swap fees or burned liquidity.

Parameters:

| Name | Type | Description |
|------------------|---------|--|
| recipient | address | The address which should receive the fees collected |
| tickLower | int24 | The lower tick of the position for which to collect fees |
| tickUpper | int24 | The upper tick of the position for which to collect fees |
| amount0Requested | uint128 | How much token0 should be withdrawn from the fees owed |
| amount1Requested | uint128 | How much token1 should be withdrawn from the fees owed |

Return Values:

| Name | Type | Description |
|---------|---------|--|
| amount0 | uint128 | The amount of fees collected in token0 |
| amount1 | uint128 | The amount of fees collected in token1 |

burn

```
function burn(
    int24 tickLower,
    int24 tickUpper,
    uint128 amount
) external override lock returns (uint256 amount0, uint256 amount1)
```

Burn liquidity from the sender and account tokens owed for the liquidity to the position

noDelegateCall is applied indirectly via _modifyPosition

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| tickLower | int24 | The lower tick of the position for which to burn liquidity |
| tickUpper | int24 | The upper tick of the position for which to burn liquidity |
| amount | uint128 | How much liquidity to burn |

Return Values:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|---------|---------|--|
| amount0 | uint256 | The amount of token0 sent to the recipient |
| amount1 | uint256 | The amount of token1 sent to the recipient |

swap

```
function swap(
    address recipient,
    bool zeroForOne,
    int256 amountSpecified,
    uint160 sqrtPriceLimitX96,
    bytes data
) external override noDelegateCall returns (int256 amount0, int256 amount1)
```

Swap token0 for token1, or token1 for token0

The caller of this method receives a callback in the form of IUniswapV3SwapCallback#uniswapV3SwapCallback

Parameters:

| Name | Type | Description |
|-------------------|---------|--|
| recipient | address | The address to receive the output of the swap |
| zeroForOne | bool | The direction of the swap, true for token0 to token1, false for token1 to token0 |
| amountSpecified | int256 | The amount of the swap, which implicitly configures the swap as exact input (positive), or exact output (negative) |
| sqrtPriceLimitX96 | uint160 | The Q64.96 sqrt price limit. If zero for one, the price cannot be less than this value after the swap. If one for zero, the price cannot be greater than this value after the swap |
| data | bytes | Any data to be passed through to the callback |

Return Values:

| Name | Type | Description |
|---------|--------|--|
| amount0 | int256 | The delta of the balance of token0 of the pool, exact when negative, minimum when positive |
| amount1 | int256 | The delta of the balance of token1 of the pool, exact when negative, minimum when positive |

flash

```
function flash(
    address recipient,
    uint256 amount0,
    uint256 amount1,
    bytes data
) external override lock noDelegateCall
```

Receive token0 and/or token1 and pay it back, plus a fee, in the callback

The caller of this method receives a callback in the form of IUniswapV3FlashCallback#uniswapV3FlashCallback Can be used to donate underlying tokens pro-rata to currently in-range liquidity providers by calling with 0 amount{0,1} and sending the donation amount(s) from the callback

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| recipient | address | The address which will receive the token0 and token1 amounts |
| amount0 | uint256 | The amount of token0 to send |
| amount1 | uint256 | The amount of token1 to send |
| data | bytes | Any data to be passed through to the callback |

setFeeProtocol

```

function setFeeProtocol(
    uint8 feeProtocol0,
    uint8 feeProtocol1
) external override lock onlyFactoryOwner

```

Set the denominator of the protocol's % share of the fees

Parameters:

| Name | Type | Description |
|--------------|-------|---|
| feeProtocol0 | uint8 | new protocol fee for token0 of the pool |
| feeProtocol1 | uint8 | new protocol fee for token1 of the pool |

collectProtocol

```

function collectProtocol(
    address recipient,
    uint128 amount0Requested,
    uint128 amount1Requested
) external override lock onlyFactoryOwner returns (uint128 amount0, uint128 amount1)

```

Collect the protocol fee accrued to the pool

Parameters:

| Name | Type | Description |
|------------------|---------|---|
| recipient | address | The address to which collected protocol fees should be sent |
| amount0Requested | uint128 | The maximum amount of token0 to send, can be 0 to collect fees in only token1 |
| amount1Requested | uint128 | The maximum amount of token1 to send, can be 0 to collect fees in only token0 |

Return Values:

| Name | Type | Description |
|---------|---------|--------------------------------------|
| amount0 | uint128 | The protocol fee collected in token0 |
| amount1 | uint128 | The protocol fee collected in token1 |

Functions

deploy

```

function deploy(
    address factory,
    address token0,
    address token1,
    uint24 fee,
    int24 tickSpacing
) internal returns (address pool)

```

Deploys a pool with the given parameters by transiently setting the parameters storage slot and then clearing it after deploying the pool.

Parameters:

| Name | Type | Description |
|-------------|---------|---|
| factory | address | The contract address of the Uniswap V3 factory |
| token0 | address | The first token of the pool by address sort order |
| token1 | address | The second token of the pool by address sort order |
| fee | uint24 | The fee collected upon every swap in the pool, denominated in hundredths of a bip |
| tickSpacing | int24 | The spacing between usable ticks |

Contains a subset of the full ERC20 interface that is used in Uniswap V3

Functions

balanceOf

```
function balanceOf(
    address account
) external view returns (uint256)
```

Returns the balance of a token

Parameters:

| Name | Type | Description |
|---------|---------|--|
| account | address | The account for which to look up the number of tokens it has, i.e. its balance |

Return Values:

| Type | Description |
|---------|--------------------------------------|
| uint256 | number of tokens held by the account |

transfer

```
function transfer(
    address recipient,
    uint256 amount
) external returns (bool)
```

Transfers the amount of token from the `msg.sender` to the recipient

Parameters:

| Name | Type | Description |
|-----------|---------|---|
| recipient | address | The account that will receive the amount transferred |
| amount | uint256 | The number of tokens to send from the sender to the recipient |

Return Values:

| Type | Description |
|------|--|
| bool | true for a successful transfer, false for an unsuccessful transfer |

allowance

```
function allowance(
    address owner,
    address spender
) external view returns (uint256)
```

Returns the current allowance given to a spender by an owner

Parameters:

| Name | Type | Description |
|---------|---------|----------------------------------|
| owner | address | The account of the token owner |
| spender | address | The account of the token spender |

Return Values:

| Type | Description |
|------|-------------|
| | |

| | |
|---------|---|
| uint256 | current allowance granted by <code>owner</code> to <code>spender</code> |
|---------|---|

approve

```
function approve(
    address spender,
    uint256 amount
) external returns (bool)
```

Sets the allowance of a spender from the `msg.sender` to the value `amount`

Parameters:

| Name | Type | Description |
|----------------------|---------|--|
| <code>spender</code> | address | The account which will be allowed to spend a given amount of the owners tokens |
| <code>amount</code> | uint256 | The amount of tokens allowed to be used by <code>spender</code> |

Return Values:

| Type | Description |
|------|--|
| bool | true for a successful approval, false for unsuccessful |

transferFrom

```
function transferFrom(
    address sender,
    address recipient,
    uint256 amount
) external returns (bool)
```

Transfers `amount` tokens from `sender` to `recipient` up to the allowance given to the `msg.sender`

Parameters:

| Name | Type | Description |
|------------------------|---------|---|
| <code>sender</code> | address | The account from which the transfer will be initiated |
| <code>recipient</code> | address | The recipient of the transfer |
| <code>amount</code> | uint256 | The amount of the transfer |

Return Values:

| Type | Description |
|------|--|
| bool | true for a successful transfer, false for unsuccessful |

Events

Transfer

```
event Transfer(
    address from,
    address to,
    uint256 value
)
```

Event emitted when tokens are transferred from one address to another, either via `#transfer` or `#transferFrom`.

Parameters:

| Name | Type | Description |
|-------------------|---------|---|
| <code>from</code> | address | The account from which the tokens were sent, i.e. the balance decreased |

| | | |
|-------|---------|---|
| to | address | The account to which the tokens were sent, i.e. the balance increased |
| value | uint256 | The amount of tokens that were transferred |

Approval

```
event Approval(
    address owner,
    address spender,
    uint256 value
)
```

Event emitted when the approval amount for the spender of a given owner's tokens changes.

Parameters:

| Name | Type | Description |
|--|---------|---|
| owner | address | The account that approved spending of its tokens |
| spender | address | The account for which the spending allowance was modified |
| value | uint256 | The new allowance from the owner to the spender |
| The Uniswap V3 Factory facilitates creation of Uniswap V3 pools and control over the protocol fees | | |

Functions

owner

```
function owner()
) external view returns (address)
```

Returns the current owner of the factory

Can be changed by the current owner via setOwner

Return Values:

| Type | Description |
|---------|------------------------------|
| address | address of the factory owner |

feeAmountTickSpacing

```
function feeAmountTickSpacing(
    uint24 fee
) external view returns (int24)
```

Returns the tick spacing for a given fee amount, if enabled, or 0 if not enabled

A fee amount can never be removed, so this value should be hard coded or cached in the calling context

Parameters:

| Name | Type | Description |
|------|--------|---|
| fee | uint24 | The enabled fee, denominated in hundredths of a bpt. Returns 0 in case of unenabled fee |

Return Values:

| Type | Description |
|-------|--------------|
| int24 | tick spacing |

getPool

```
function getPool(
    address tokenA,
    address tokenB,
    uint24 fee
) external view returns (address pool)
```

Returns the pool address for a given pair of tokens and a fee, or address 0 if it does not exist

tokenA and tokenB may be passed in either token0/token1 or token1/token0 order

Parameters:

| Name | Type | Description |
|--------|---------|---|
| tokenA | address | The contract address of either token0 or token1 |
| tokenB | address | The contract address of the other token |
| fee | uint24 | The fee collected upon every swap in the pool, denominated in hundredths of a bip |

Return Values:

| Name | Type | Description |
|------|---------|------------------|
| pool | address | The pool address |

createPool

```
function createPool(
    address tokenA,
    address tokenB,
    uint24 fee
) external returns (address pool)
```

Creates a pool for the given two tokens and fee

tokenA and tokenB may be passed in either order: token0/token1 or token1/token0. tickSpacing is retrieved from the fee. The call will revert if the pool already exists, the fee is invalid, or the token arguments are invalid.

Parameters:

| Name | Type | Description |
|--------|---------|---|
| tokenA | address | One of the two tokens in the desired pool |
| tokenB | address | The other of the two tokens in the desired pool |
| fee | uint24 | The desired fee for the pool |

Return Values:

| Name | Type | Description |
|------|---------|---------------------------------------|
| pool | address | The address of the newly created pool |

setOwner

```
function setOwner(
    address _owner
) external
```

Updates the owner of the factory

Must be called by the current owner

Parameters:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|---------------------|----------------------|------------------------------|
| <code>_owner</code> | <code>address</code> | The new owner of the factory |
|---------------------|----------------------|------------------------------|

enableFeeAmount

```
function enableFeeAmount(
    uint24 fee,
    int24 tickSpacing
) external
```

Enables a fee amount with the given tickSpacing

Fee amounts may never be removed once enabled

Parameters:

| Name | Type | Description |
|--------------------------|---------------------|--|
| <code>fee</code> | <code>uint24</code> | The fee amount to enable, denominated in hundredths of a bip (i.e. 1e-6) |
| <code>tickSpacing</code> | <code>int24</code> | The spacing between ticks to be enforced for all pools created with the given fee amount |

Events

OwnerChanged

```
event OwnerChanged(
    address oldOwner,
    address newOwner
)
```

Emitted when the owner of the factory is changed

Parameters:

| Name | Type | Description |
|-----------------------|----------------------|--|
| <code>oldOwner</code> | <code>address</code> | The owner before the owner was changed |
| <code>newOwner</code> | <code>address</code> | The owner after the owner was changed |

PoolCreated

```
event PoolCreated(
    address token0,
    address token1,
    uint24 fee,
    int24 tickSpacing,
    address pool
)
```

Emitted when a pool is created

Parameters:

| Name | Type | Description |
|--------------------------|----------------------|---|
| <code>token0</code> | <code>address</code> | The first token of the pool by address sort order |
| <code>token1</code> | <code>address</code> | The second token of the pool by address sort order |
| <code>fee</code> | <code>uint24</code> | The fee collected upon every swap in the pool, denominated in hundredths of a bip |
| <code>tickSpacing</code> | <code>int24</code> | The minimum number of ticks between initialized ticks |
| <code>pool</code> | <code>address</code> | The address of the created pool |

FeeAmountEnabled

```

event FeeAmountEnabled(
    uint24 fee,
    int24 tickSpacing
)

```

Emitted when a new fee amount is enabled for pool creation via the factory

Parameters:

| Name | Type | Description |
|---|--------|--|
| fee | uint24 | The enabled fee, denominated in hundredths of a bip |
| tickSpacing | int24 | The minimum number of ticks between initialized ticks for pools created with the given fee |
| A Uniswap pool facilitates swapping and automated market making between any two assets that strictly conform to the ERC20 specification | | |

The pool interface is broken up into many smaller pieces A contract that constructs a pool must implement this to pass arguments to the pool

This is used to avoid having constructor arguments in the pool contract, which results in the init code hash of the pool being constant allowing the CREATE2 address of the pool to be cheaply computed on-chain

Functions

parameters

```

function parameters(
) external view returns (address factory, address token0, address token1, uint24 fee, int24 tickSpacing)

```

Get the parameters to be used in constructing the pool, set transiently during pool creation.

Return Values :

| Name | Type | Description |
|---|---------|---|
| factory | address | The factory address |
| token0 | address | The first token of the pool by address sort order |
| token1 | address | The second token of the pool by address sort order |
| fee | uint24 | The fee collected upon every swap in the pool, denominated in hundredths of a bip |
| tickSpacing | int24 | The minimum number of ticks between initialized ticks |
| Any contract that calls IUniswapV3PoolActions#flash must implement this interface | | |

Functions

uniswapV3FlashCallback

```

function uniswapV3FlashCallback(
    uint256 fee0,
    uint256 fee1,
    bytes data
) external

```

Called to `msg.sender` after transferring to the recipient from IUniswapV3Pool#flash.

In the implementation you must repay the pool the tokens sent by flash plus the computed fee amounts. The caller of this method must be checked to be a UniswapV3Pool deployed by the canonical UniswapV3Factory.

Parameters:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|--|---------|--|
| fee0 | uint256 | The fee amount in token0 due to the pool by the end of the flash |
| fee1 | uint256 | The fee amount in token1 due to the pool by the end of the flash |
| data | bytes | Any data passed through by the caller via the IUniswapV3PoolActions#flash call |
| Any contract that calls IUniswapV3PoolActions#mint must implement this interface | | |

Functions

uniswapV3MintCallback

```
function uniswapV3MintCallback(
    uint256 amount0Owed,
    uint256 amount1Owed,
    bytes data
) external
```

Called to `msg.sender` after minting liquidity to a position from IUniswapV3Pool#mint.

In the implementation you must pay the pool tokens owed for the minted liquidity. The caller of this method must be checked to be a UniswapV3Pool deployed by the canonical UniswapV3Factory.

Parameters:

| Name | Type | Description |
|--|---------|---|
| amount0Owed | uint256 | The amount of token0 due to the pool for the minted liquidity |
| amount1Owed | uint256 | The amount of token1 due to the pool for the minted liquidity |
| data | bytes | Any data passed through by the caller via the IUniswapV3PoolActions#mint call |
| Any contract that calls IUniswapV3PoolActions#swap must implement this interface | | |

Functions

uniswapV3SwapCallback

```
function uniswapV3SwapCallback(
    int256 amount0Delta,
    int256 amount1Delta,
    bytes data
) external
```

Called to `msg.sender` after executing a swap via IUniswapV3Pool#swap.

In the implementation you must pay the pool tokens owed for the swap. The caller of this method must be checked to be a UniswapV3Pool deployed by the canonical UniswapV3Factory. `amount0Delta` and `amount1Delta` can both be 0 if no tokens were swapped.

Parameters:

| Name | Type | Description |
|--|--------|---|
| amount0Delta | int256 | The amount of token0 that was sent (negative) or must be received (positive) by the pool by the end of the swap. If positive, the callback must send that amount of token0 to the pool. |
| amount1Delta | int256 | The amount of token1 that was sent (negative) or must be received (positive) by the pool by the end of the swap. If positive, the callback must send that amount of token1 to the pool. |
| data | bytes | Any data passed through by the caller via the IUniswapV3PoolActions#swap call. |
| Contains pool methods that can be called by anyone | | |

Functions

initialize

```
function initialize(
    uint160 sqrtPriceX96
) external
```

Sets the initial price for the pool

Price is represented as a $\text{sqrt}(\text{amountToken1}/\text{amountToken0})$ Q64.96 value

Parameters:

| Name | Type | Description |
|--------------|---------|--|
| sqrtPriceX96 | uint160 | the initial sqrt price of the pool as a Q64.96 |

mint

```
function mint(
    address recipient,
    int24 tickLower,
    int24 tickUpper,
    uint128 amount,
    bytes data
) external returns (uint256 amount0, uint256 amount1)
```

Adds liquidity for the given recipient/tickLower/tickUpper position

The caller of this method receives a callback in the form of IUniswapV3MintCallback#uniswapV3MintCallback in which they must pay any token0 or token1 owed for the liquidity. The amount of token0/token1 due depends on tickLower, tickUpper, the amount of liquidity, and the current price.

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| recipient | address | The address for which the liquidity will be created |
| tickLower | int24 | The lower tick of the position in which to add liquidity |
| tickUpper | int24 | The upper tick of the position in which to add liquidity |
| amount | uint128 | The amount of liquidity to mint |
| data | bytes | Any data that should be passed through to the callback |

Return Values:

| Name | Type | Description |
|---------|---------|---|
| amount0 | uint256 | The amount of token0 that was paid to mint the given amount of liquidity. Matches the value in the callback |
| amount1 | uint256 | The amount of token1 that was paid to mint the given amount of liquidity. Matches the value in the callback |

collect

```
function collect(
    address recipient,
    int24 tickLower,
    int24 tickUpper,
    uint128 amount0Requested,
    uint128 amount1Requested
) external returns (uint128 amount0, uint128 amount1)
```

Collects tokens owed to a position

Does not recompute fees earned, which must be done either via mint or burn of any amount of liquidity. Collect must be called by the position owner. To withdraw only token0 or only token1, amount0Requested or amount1Requested may be set to zero. To withdraw all tokens owed, caller

may pass any value greater than the actual tokens owed, e.g. type(uint128).max. Tokens owed may be from accumulated swap fees or burned liquidity.

Parameters:

| Name | Type | Description |
|------------------|---------|--|
| recipient | address | The address which should receive the fees collected |
| tickLower | int24 | The lower tick of the position for which to collect fees |
| tickUpper | int24 | The upper tick of the position for which to collect fees |
| amount0Requested | uint128 | How much token0 should be withdrawn from the fees owed |
| amount1Requested | uint128 | How much token1 should be withdrawn from the fees owed |

Return Values:

| Name | Type | Description |
|---------|---------|--|
| amount0 | uint128 | The amount of fees collected in token0 |
| amount1 | uint128 | The amount of fees collected in token1 |

burn

```
function burn(
    int24 tickLower,
    int24 tickUpper,
    uint128 amount
) external returns (uint256 amount0, uint256 amount1)
```

Burn liquidity from the sender and account tokens owed for the liquidity to the position

Can be used to trigger a recalculation of fees owed to a position by calling with an amount of 0 Fees must be collected separately via a call to #collect

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| tickLower | int24 | The lower tick of the position for which to burn liquidity |
| tickUpper | int24 | The upper tick of the position for which to burn liquidity |
| amount | uint128 | How much liquidity to burn |

Return Values:

| Name | Type | Description |
|---------|---------|--|
| amount0 | uint256 | The amount of token0 sent to the recipient |
| amount1 | uint256 | The amount of token1 sent to the recipient |

swap

```
function swap(
    address recipient,
    bool zeroForOne,
    int256 amountSpecified,
    uint160 sqrtPriceLimitX96,
    bytes data
) external returns (int256 amount0, int256 amount1)
```

Swap token0 for token1, or token1 for token0

The caller of this method receives a callback in the form of IUniswapV3SwapCallback#uniswapV3SwapCallback

Parameters:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|-------------------|---------|--|
| recipient | address | The address to receive the output of the swap |
| zeroForOne | bool | The direction of the swap, true for token0 to token1, false for token1 to token0 |
| amountSpecified | int256 | The amount of the swap, which implicitly configures the swap as exact input (positive), or exact output (negative) |
| sqrtPriceLimitX96 | uint160 | The Q64.96 sqrt price limit. If zero for one, the price cannot be less than this value after the swap. If one for zero, the price cannot be greater than this value after the swap |
| data | bytes | Any data passed through to the callback |

Return Values:

| Name | Type | Description |
|---------|--------|--|
| amount0 | int256 | The delta of the balance of token0 of the pool, exact when negative, minimum when positive |
| amount1 | int256 | The delta of the balance of token1 of the pool, exact when negative, minimum when positive |

flash

```
function flash(
    address recipient,
    uint256 amount0,
    uint256 amount1,
    bytes data
) external
```

Receive token0 and/or token1 and pay it back, plus a fee, in the callback

The caller of this method receives a callback in the form of IUniswapV3FlashCallback#uniswapV3FlashCallback Can be used to donate underlying tokens pro-rata to currently in-range liquidity providers by calling with 0 amount{0,1} and sending the donation amount(s) from the callback

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| recipient | address | The address which will receive the token0 and token1 amounts |
| amount0 | uint256 | The amount of token0 to send |
| amount1 | uint256 | The amount of token1 to send |
| data | bytes | Any data to be passed through to the callback |

increaseObservationCardinalityNext

```
function increaseObservationCardinalityNext(
    uint16 observationCardinalityNext
) external
```

Increase the maximum number of price and liquidity observations that this pool will store

This method is no-op if the pool already has an observationCardinalityNext greater than or equal to the input observationCardinalityNext.

Parameters:

| Name | Type | Description |
|--|--------|--|
| observationCardinalityNext | uint16 | The desired minimum number of observations for the pool to store |
| Contains view functions to provide information about the pool that is computed rather than stored on the blockchain. The functions here may have variable gas costs. | | |
| | | |

Functions

observe

```

function observe(
    uint32[] secondsAgo
) external view returns (int56[] tickCumulatives, uint160[] secondsPerLiquidityCumulativeX128s)

```

Returns the cumulative tick and liquidity as of each timestamp `secondsAgo` from the current block timestamp

To get a time weighted average tick or liquidity-in-range, you must call this with two values, one representing the beginning of the period and another for the end of the period. E.g., to get the last hour time-weighted average tick, you must call it with `secondsAgo = [3600, 0]`. The time weighted average tick represents the geometric time weighted average price of the pool, in log base $\sqrt{1.0001}$ of `token1 / token0`. The `TickMath` library can be used to go from a tick value to a ratio.

Parameters:

| Name | Type | Description |
|-------------------------|-----------------------|---|
| <code>secondsAgo</code> | <code>uint32[]</code> | From how long ago each cumulative tick and liquidity value should be returned |

Return Values:

| Name | Type | Description |
|---|------------------------|---|
| <code>tickCumulatives</code> | <code>int56[]</code> | Cumulative tick values as of each <code>secondsAgo</code> from the current block timestamp |
| <code>secondsPerLiquidityCumulativeX128s</code> | <code>uint160[]</code> | Cumulative seconds per liquidity-in-range value as of each <code>secondsAgo</code> from the current block |

`timestamp`

snapshotCumulativesInside

```

function snapshotCumulativesInside(
    int24 tickLower,
    int24 tickUpper
) external returns (int56 tickCumulativeInside, uint160 secondsPerLiquidityInsideX128, uint32 secondsInside)

```

Returns a snapshot of the tick cumulative, seconds per liquidity and seconds inside a tick range

Snapshots must only be compared to other snapshots, taken over a period for which a position existed. I.e., snapshots cannot be compared if a position is not held for the entire period between when the first snapshot is taken and the second snapshot is taken.

Parameters:

| Name | Type | Description |
|------------------------|--------------------|-----------------------------|
| <code>tickLower</code> | <code>int24</code> | The lower tick of the range |
| <code>tickUpper</code> | <code>int24</code> | The upper tick of the range |

Return Values:

| Name | Type | Description |
|--|----------------------|---|
| <code>tickCumulativeInside</code> | <code>int56</code> | The snapshot of the tick accumulator for the range |
| <code>secondsPerLiquidityInsideX128</code> | <code>uint160</code> | The snapshot of seconds per liquidity for the range |
| <code>secondsInside</code> | <code>uint32</code> | The snapshot of seconds per liquidity for the range |
| Contains all events emitted by the pool | | |

Events

Initialize

```

event Initialize(
    uint160 sqrtPriceX96,
    int24 tick
)

```

Emitted exactly once by a pool when `#initialize` is first called on the pool

Mint/Burn/Swap cannot be emitted by the pool before Initialize

Parameters:

| Name | Type | Description |
|--------------|---------|--|
| sqrtPriceX96 | uint160 | The initial sqrt price of the pool, as a Q64.96 |
| tick | int24 | The initial tick of the pool, i.e. log base 1.0001 of the starting price of the pool |

Mint

```
event Mint(
    address sender,
    address owner,
    int24 tickLower,
    int24 tickUpper,
    uint128 amount,
    uint256 amount0,
    uint256 amount1
)
```

Emitted when liquidity is minted for a given position

Parameters:

| Name | Type | Description |
|-----------|---------|---|
| sender | address | The address that minted the liquidity |
| owner | address | The owner of the position and recipient of any minted liquidity |
| tickLower | int24 | The lower tick of the position |
| tickUpper | int24 | The upper tick of the position |
| amount | uint128 | The amount of liquidity minted to the position range |
| amount0 | uint256 | How much token0 was required for the minted liquidity |
| amount1 | uint256 | How much token1 was required for the minted liquidity |

Collect

```
event Collect(
    address owner,
    int24 tickLower,
    int24 tickUpper,
    uint128 amount0,
    uint128 amount1
)
```

Emitted when fees are collected by the owner of a position

Collect events may be emitted with zero amount0 and amount1 when the caller chooses not to collect fees

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| owner | address | The owner of the position for which fees are collected |
| tickLower | int24 | The lower tick of the position |
| tickUpper | int24 | The upper tick of the position |
| amount0 | uint128 | The amount of token0 fees collected |
| amount1 | uint128 | The amount of token1 fees collected |

Burn

```

event Burn(
    address owner,
    int24 tickLower,
    int24 tickUpper,
    uint128 amount,
    uint256 amount0,
    uint256 amount1
)

```

Emitted when a position's liquidity is removed

Does not withdraw any fees earned by the liquidity position, which must be withdrawn via #collect

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| owner | address | The owner of the position for which liquidity is removed |
| tickLower | int24 | The lower tick of the position |
| tickUpper | int24 | The upper tick of the position |
| amount | uint128 | The amount of liquidity to remove |
| amount0 | uint256 | The amount of token0 withdrawn |
| amount1 | uint256 | The amount of token1 withdrawn |

Swap

```

event Swap(
    address sender,
    address recipient,
    int256 amount0,
    int256 amount1,
    uint160 sqrtPriceX96,
    uint128 liquidity,
    int24 tick
)

```

Emitted by the pool for any swaps between token0 and token1

Parameters:

| Name | Type | Description |
|--------------|---------|--|
| sender | address | The address that initiated the swap call, and that received the callback |
| recipient | address | The address that received the output of the swap |
| amount0 | int256 | The delta of the token0 balance of the pool |
| amount1 | int256 | The delta of the token1 balance of the pool |
| sqrtPriceX96 | uint160 | The sqrt(price) of the pool after the swap, as a Q64.96 |
| liquidity | uint128 | The liquidity of the pool after the swap |
| tick | int24 | The log base 1.0001 of price of the pool after the swap |

Flash

```

event Flash(
    address sender,
    address recipient,
    uint256 amount0,
    uint256 amount1,
    uint256 paid0,
    uint256 paid1
)

```

Emitted by the pool for any flashes of token0/token1

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| sender | address | The address that initiated the swap call, and that received the callback |
| recipient | address | The address that received the tokens from flash |
| amount0 | uint256 | The amount of token0 that was flashed |
| amount1 | uint256 | The amount of token1 that was flashed |
| paid0 | uint256 | The amount of token0 paid for the flash, which can exceed the amount0 plus the fee |
| paid1 | uint256 | The amount of token1 paid for the flash, which can exceed the amount1 plus the fee |

IncreaseObservationCardinalityNext

```
event IncreaseObservationCardinalityNext(
    uint16 observationCardinalityNextOld,
    uint16 observationCardinalityNextNew
)
```

Emitted by the pool for increases to the number of observations that can be stored

observationCardinalityNext is not the observation cardinality until an observation is written at the index just before a mint/swap/burn.

Parameters:

| Name | Type | Description |
|-------------------------------|--------|--|
| observationCardinalityNextOld | uint16 | The previous value of the next observation cardinality |
| observationCardinalityNextNew | uint16 | The updated value of the next observation cardinality |

SetFeeProtocol

```
event SetFeeProtocol(
    uint8 feeProtocol0Old,
    uint8 feeProtocol1Old,
    uint8 feeProtocol0New,
    uint8 feeProtocol1New
)
```

Emitted when the protocol fee is changed by the pool

Parameters:

| Name | Type | Description |
|-----------------|-------|---|
| feeProtocol0Old | uint8 | The previous value of the token0 protocol fee |
| feeProtocol1Old | uint8 | The previous value of the token1 protocol fee |
| feeProtocol0New | uint8 | The updated value of the token0 protocol fee |
| feeProtocol1New | uint8 | The updated value of the token1 protocol fee |

CollectProtocol

```
event CollectProtocol(
    address sender,
    address recipient,
    uint128 amount0,
    uint128 amount1
)
```

Emitted when the collected protocol fees are withdrawn by the factory owner

Parameters:

| Name | Type | Description |
|---|---------|---|
| sender | address | The address that collects the protocol fees |
| recipient | address | The address that receives the collected protocol fees |
| amount0 | uint128 | The amount of token0 protocol fees that is withdrawn |
| amount1 | uint128 | The amount of token1 protocol fees that is withdrawn |
| These parameters are fixed for a pool forever, i.e., the methods will always return the same values | | |

Functions

factory

```
function factory(
) external view returns (address)
```

The contract that deployed the pool, which must adhere to the IUniswapV3Factory interface

Return Values:

| Type | Description |
|---------|------------------|
| address | contract address |

token0

```
function token0(
) external view returns (address)
```

The first of the two tokens of the pool, sorted by address

Return Values:

| Type | Description |
|---------|------------------------|
| address | token contract address |

token1

```
function token1(
) external view returns (address)
```

The second of the two tokens of the pool, sorted by address

Return Values:

| Type | Description |
|---------|------------------------|
| address | token contract address |

fee

```
function fee(
) external view returns (uint24)
```

The pool's fee in hundredths of a bip, i.e. 1e-6

Return Values:

| Type | Description |
|------|-------------|
| | |

| | |
|--------|-----|
| uint24 | fee |
|--------|-----|

tickSpacing

```
function tickSpacing(
) external view returns (int24)
```

The pool tick spacing

Ticks can only be used at multiples of this value, minimum of 1 and always positive e.g.: a tickSpacing of 3 means ticks can be initialized every 3rd tick, i.e., ..., -6, -3, 0, 3, 6, ... This value is an int24 to avoid casting even though it is always positive.

Return Values:

| Type | Description |
|-------|--------------|
| int24 | tick spacing |

maxLiquidityPerTick

```
function maxLiquidityPerTick(
) external view returns (uint128)
```

The maximum amount of position liquidity that can use any tick in the range

This parameter is enforced per tick to prevent liquidity from overflowing a uint128 at any point, and also prevents out-of-range liquidity from being used to prevent adding in-range liquidity to a pool

Return Values:

| Type | Description |
|--|----------------------------------|
| uint128 | max amount of liquidity per tick |
| Contains pool methods that may only be called by the factory owner | |

Functions

setFeeProtocol

```
function setFeeProtocol(
    uint8 feeProtocol0,
    uint8 feeProtocol1
) external
```

Set the denominator of the protocol's % share of the fees

Parameters:

| Name | Type | Description |
|--------------|-------|---|
| feeProtocol0 | uint8 | new protocol fee for token0 of the pool |
| feeProtocol1 | uint8 | new protocol fee for token1 of the pool |

collectProtocol

```
function collectProtocol(
    address recipient,
    uint128 amount0Requested,
    uint128 amount1Requested
) external returns (uint128 amount0, uint128 amount1)
```

Collect the protocol fee accrued to the pool

Parameters:

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|------------------|---------|---|
| recipient | address | The address to which collected protocol fees should be sent |
| amount0Requested | uint128 | The maximum amount of token0 to send, can be 0 to collect fees in only token1 |
| amount1Requested | uint128 | The maximum amount of token1 to send, can be 0 to collect fees in only token0 |

Return Values:

| Name | Type | Description |
|--|---------|--------------------------------------|
| amount0 | uint128 | The protocol fee collected in token0 |
| amount1 | uint128 | The protocol fee collected in token1 |
| These methods compose the pool's state, and can change with any frequency including multiple times | | |
| per transaction | | |

Functions

slot0

```
function slot0()
) external view returns (uint160 sqrtPriceX96, int24 tick, uint16 observationIndex, uint16
observationCardinality, uint16 observationCardinalityNext, uint8 feeProtocol, bool unlocked)
```

The 0th storage slot in the pool stores many values, and is exposed as a single method to save gas when accessed externally.

Return Values:

| Name | Type | Description |
|----------------------------|---------|--|
| sqrtPriceX96 | uint160 | The current price of the pool as a sqrt(token1/token0) Q64.96 value |
| tick | int24 | The current tick of the pool, i.e. according to the last tick transition that was run. This value may not always be equal to SqrtTickMath.getTickAtSqrRatio(sqrtPriceX96) if the price is on a tick boundary. |
| observationIndex | uint16 | The index of the last oracle observation that was written, |
| observationCardinality | uint16 | The current maximum number of observations stored in the pool, |
| observationCardinalityNext | uint16 | The next maximum number of observations, to be updated when the observation. |
| feeProtocol | uint8 | The protocol fee for both tokens of the pool. Encoded as two 4 bit values, where the protocol fee of token1 is shifted 4 bits and the protocol fee of token0 is the lower 4 bits. Used as the denominator of a fraction of the swap fee, e.g. 4 means 1/4th of the swap fee. |
| unlocked | bool | Whether the pool is currently locked to reentrancy |

feeGrowthGlobal0X128

```
function feeGrowthGlobal0X128(
) external view returns (uint256)
```

The fee growth as a Q128.128 fees of token0 collected per unit of liquidity for the entire life of the pool

This value can overflow the uint256

feeGrowthGlobal1X128

```
function feeGrowthGlobal1X128(
) external view returns (uint256)
```

The fee growth as a Q128.128 fees of token1 collected per unit of liquidity for the entire life of the pool

This value can overflow the uint256

protocolFees

```
function protocolFees()
) external view returns (uint128 token0, uint128 token1)
```

The amounts of token0 and token1 that are owed to the protocol

Protocol fees will never exceed uint128 max in either token

liquidity

```
function liquidity(
) external view returns (uint128)
```

The currently in range liquidity available to the pool

This value has no relationship to the total liquidity across all ticks

ticks

```
function ticks(
int24 tick
) external view returns (uint128 liquidityGross, int128 liquidityNet, uint256 feeGrowthOutside0X128, uint256
feeGrowthOutside1X128, int56 tickCumulativeOutside, uint160 secondsPerLiquidityOutsideX128, uint32 secondsOutside,
bool initialized)
```

Look up information about a specific tick in the pool

Parameters:

| Name | Type | Description |
|------|-------|---------------------|
| tick | int24 | The tick to look up |

Return Values:

| Name | Type | Description |
|--------------------------------|---------|---|
| liquidityGross | uint128 | the total amount of position liquidity that uses the pool either as tick lower or tick upper, |
| liquidityNet | int128 | how much liquidity changes when the pool price crosses the tick, |
| feeGrowthOutside0X128 | uint256 | the fee growth on the other side of the tick from the current tick in token0, |
| feeGrowthOutside1X128 | uint256 | the fee growth on the other side of the tick from the current tick in token1,
feeGrowthOutsideX128 values can only be used if the tick is initialized, i.e. if liquidityGross is
greater than 0. In addition, these values are only relative and are used to compute snapshots. |
| tickCumulativeOutside | int56 | |
| secondsPerLiquidityOutsideX128 | uint160 | |
| secondsOutside | uint32 | |
| initialized | bool | |

a specific position.

tickBitmap

```
function tickBitmap(
int16 wordPosition
) external view returns (uint256)
```

Returns 256 packed tick initialized boolean values. See TickBitmap for more information

positions

```

function positions(
    bytes32 key
) external view returns (uint128 _liquidity, uint256 feeGrowthInside0LastX128, uint256 feeGrowthInside1LastX128,
    uint128 tokensOwed0, uint128 tokensOwed1)

```

Returns the information about a position by the position's key

Parameters:

| Name | Type | Description |
|------|---------|---|
| key | bytes32 | The position's key is a hash of a preimage composed by the owner, tickLower and tickUpper |

Return Values:

| Name | Type | Description |
|--------------------------|---------|---|
| _liquidity | bytes32 | The amount of liquidity in the position, |
| feeGrowthInside0LastX128 | uint256 | fee growth of token0 inside the tick range as of the last mint/burn/poke, |
| feeGrowthInside1LastX128 | uint256 | fee growth of token1 inside the tick range as of the last mint/burn/poke, |
| tokensOwed0 | uint128 | the computed amount of token0 owed to the position as of the last mint/burn/poke, |
| tokensOwed1 | uint128 | the computed amount of token1 owed to the position as of the last mint/burn/poke |

Observations

```

function observations(
    uint256 index
) external view returns (uint32 blockTimestamp, int56 tickCumulative, uint160 secondsPerLiquidityCumulativeX128,
bool initialized)

```

Returns data about a specific observation index

You most likely want to use `#observe()` instead of this method to get an observation as of some amount of time ago, rather than at a specific index in the array.

Parameters:

| Name | Type | Description |
|-------|---------|--|
| index | uint256 | The element of the observations array to fetch |

Return Values:

| Name | Type | Description |
|---|---------|--|
| blockTimestamp | uint256 | The timestamp of the observation, |
| tickCumulative | int56 | the tick multiplied by seconds elapsed for the life of the pool as of the observation timestamp, |
| secondsPerLiquidityCumulativeX128 | uint160 | the seconds per in range liquidity for the life of the pool as of the observation timestamp |
| initialized | bool | whether the observation has been initialized and the values are safe to use |
| This library provides functionality for computing bit properties of an unsigned integer | | |

Functions

mostSignificantBit

```

function mostSignificantBit(
    uint256 x
) internal pure returns (uint8 r)

```

Returns the index of the most significant bit of the number, where the least significant bit is at index 0 and the most significant bit is at index 255

The function satisfies the property: $x \geq 2^{\text{mostSignificantBit}(x)}$ and $x < 2^{\text{mostSignificantBit}(x)+1}$

Parameters:

| Name | Type | Description |
|------|---------|---|
| x | uint256 | the value for which to compute the most significant bit, must be greater than 0 |

Return Values:

| Name | Type | Description |
|------|-------|---------------------------------------|
| r | unit8 | the index of the most significant bit |

leastSignificantBit

```
function leastSignificantBit(
    uint256 x
) internal pure returns (uint8 r)
```

Returns the index of the least significant bit of the number, where the least significant bit is at index 0 and the most significant bit is at index 255

The function satisfies the property: $(x \& 2^{\text{leastSignificantBit}(x)}) \neq 0$ and $(x \& (2^{\text{leastSignificantBit}(x)} - 1)) == 0$

Parameters:

| Name | Type | Description |
|------|---------|--|
| x | uint256 | the value for which to compute the least significant bit, must be greater than 0 |

Return Values:

| Name | Type | Description |
|---|-------|--|
| r | unit8 | the index of the least significant bit |
| A library for handling binary fixed point numbers, see
https://en.wikipedia.org/wiki/Q_(number_format) | | |
| A library for handling binary fixed point numbers, see
https://en.wikipedia.org/wiki/Q_(number_format) | | |

Used in SqrtPriceMath.sol Facilitates multiplication and division that can have overflow of an intermediate value without any loss of precision

Handles "phantom overflow" i.e., allows multiplication and division where an intermediate value overflows 256 bits

Functions

mulDiv

```
function mulDiv(
    uint256 a,
    uint256 b,
    uint256 denominator
) internal pure returns (uint256 result)
```

Calculates $\text{floor}(a \times b / \text{denominator})$ with full precision. Throws if result overflows a uint256 or denominator == 0

Credit to Remco Bloemen under MIT license <https://xn--2-umb.com/21/muldiv>

Parameters:

| Name | Type | Description |
|-------------|---------|------------------|
| a | uint256 | The multiplicand |
| b | uint256 | The multiplier |
| denominator | uint256 | The divisor |

Return Values:

| Name | Type | Description |
|--------|---------|--------------------|
| result | uint256 | The 256-bit result |

mulDivRoundingUp

```
function mulDivRoundingUp(
    uint256 a,
    uint256 b,
    uint256 denominator
) internal pure returns (uint256 result)
```

Calculates $\text{ceil}(axb \div \text{denominator})$ with full precision. Throws if result overflows a uint256 or denominator == 0

Parameters:

| Name | Type | Description |
|-------------|---------|------------------|
| a | uint256 | The multiplicand |
| b | uint256 | The multiplier |
| denominator | uint256 | The divisor |

Return Values:

| Name | Type | Description |
|--------|---------|--------------------|
| result | uint256 | The 256-bit result |

Functions**addDelta**

```
function addDelta(
    uint128 x,
    int128 y
) internal pure returns (uint128 z)
```

Add a signed liquidity delta to liquidity and revert if it overflows or underflows

Parameters:

| Name | Type | Description |
|------|---------|--|
| x | uint128 | The liquidity before change |
| y | int128 | The delta by which liquidity should be changed |

Return Values:

| Name | Type | Description |
|--|---------|---------------------|
| z | uint128 | The liquidity delta |
| Contains methods for doing math operations that revert on overflow or underflow for minimal gas cost | | |

Functions**add**

```
function add(
    uint256 x,
    uint256 y
) internal pure returns (uint256 z)
```

Returns $x + y$, reverts if sum overflows uint256

Parameters:

| Name | Type | Description |
|------|---------|-------------|
| x | uint256 | The augend |
| y | uint256 | The addend |

Return Values:

| Name | Type | Description |
|------|---------|--------------------|
| z | uint256 | The sum of x and y |

sub

```
function sub(
    uint256 x,
    uint256 y
) internal pure returns (uint256 z)
```

Returns x - y, reverts if underflows

Parameters:

| Name | Type | Description |
|------|---------|----------------|
| x | uint256 | The minuend |
| y | uint256 | The subtrahend |

Return Values:

| Name | Type | Description |
|------|---------|---------------------------|
| z | uint256 | The difference of x and y |

mul

```
function mul(
    uint256 x,
    uint256 y
) internal pure returns (uint256 z)
```

Returns x * y, reverts if overflows

Parameters:

| Name | Type | Description |
|------|---------|------------------|
| x | uint256 | The multiplicand |
| y | uint256 | The multiplier |

Return Values:

| Name | Type | Description |
|------|---------|------------------------|
| z | uint256 | The product of x and y |

add

```
function add(
    int256 x,
    int256 y
) internal pure returns (int256 z)
```

Returns x + y, reverts if overflows or underflows

Parameters:

| Name | Type | Description |
|------|--------|-------------|
| x | int256 | The augend |
| y | int256 | The addend |

Return Values:

| Name | Type | Description |
|------|--------|--------------------|
| z | int256 | The sum of x and y |

sub

```
function sub(
    int256 x,
    int256 y
) internal pure returns (int256 z)
```

Returns x - y, reverts if overflows or underflows

Parameters:

| Name | Type | Description |
|------|--------|----------------|
| x | int256 | The minuend |
| y | int256 | The subtrahend |

Return Values:

| Name | Type | Description |
|---|--------|---------------------------|
| z | int256 | The difference of x and y |
| Provides price and liquidity data useful for a wide variety of system designs | | |

Instances of stored oracle data, "observations", are collected in the oracle array. Every pool is initialized with an oracle array length of 1. Anyone can pay the SSTOREs to increase the maximum length of the oracle array. New slots will be added when the array is fully populated. Observations are overwritten when the full length of the oracle array is populated. The most recent observation is available, independent of the length of the oracle array, by passing 0 to observe()

Functions

initialize

```
function initialize(
    struct Oracle.Observation[65535] self,
    uint32 time
) internal returns (uint16 cardinality, uint16 cardinalityNext)
```

Initialize the oracle array by writing the first slot. Called once for the lifecycle of the observations array

Parameters:

| Name | Type | Description |
|------|----------------------------------|--|
| self | struct Oracle.Observation[65535] | The stored oracle array |
| time | uint32 | The time of the oracle initialization, via block.timestamp truncated to uint32 |

Return Values:

| Name | Type | Description |
|-----------------|--------|---|
| cardinality | uint16 | The number of populated elements in the oracle array |
| cardinalityNext | uint16 | The new length of the oracle array, independent of population |

write

```

function write(
    struct Oracle.Observation[65535] self,
    uint16 index,
    uint32 blockTimestamp,
    int24 tick,
    uint128 liquidity,
    uint16 cardinality,
    uint16 cardinalityNext
) internal returns (uint16 indexUpdated, uint16 cardinalityUpdated)

```

Writes an oracle observation to the array

Writable at most once per block. Index represents the most recently written element. cardinality and index must be tracked externally. If the index is at the end of the allowable array length (according to cardinality), and the next cardinality is greater than the current one, cardinality may be increased. This restriction is created to preserve ordering.

Parameters:

| Name | Type | Description |
|-----------------|----------------------------------|---|
| self | struct Oracle.Observation[65535] | The stored oracle array |
| index | uint16 | The location of the most recently updated observation |
| blockTimestamp | uint32 | The timestamp of the new observation |
| tick | int24 | The active tick at the time of the new observation |
| liquidity | uint128 | The total in-range liquidity at the time of the new observation |
| cardinality | uint16 | The number of populated elements in the oracle array |
| cardinalityNext | uint16 | The new length of the oracle array, independent of population |

Return Values:

| Name | Type | Description |
|--------------------|--------|--|
| indexUpdated | uint16 | The new index of the most recently written element in the oracle array |
| cardinalityUpdated | uint16 | The new cardinality of the oracle array |

grow

```

function grow(
    struct Oracle.Observation[65535] self,
    uint16 current,
    uint16 next
) internal returns (uint16)

```

Prepares the oracle array to store up to `next` observations

Parameters:

| Name | Type | Description |
|---------|----------------------------------|---|
| self | struct Oracle.Observation[65535] | The stored oracle array |
| current | uint16 | The current next cardinality of the oracle array |
| next | uint16 | The proposed next cardinality which will be populated in the oracle array |

Return Values:

| Name | Type | Description |
|------|--------|--|
| next | uint16 | The next cardinality which will be populated in the oracle array |

observe

```

function observe(
    struct Oracle.Observation[65535] self,
    uint32 time,
    uint32[] secondsAgo,
    int24 tick,
    uint16 index,
    uint128 liquidity,
    uint16 cardinality
) internal view returns (int56[] tickCumulatives, uint160[] liquidityCumulatives)

```

Returns the accumulator values as of each time seconds ago from the given time in the array of `secondsAgo`

Reverts if `secondsAgo > oldest observation`

Parameters:

| Name | Type | Description |
|--------------------------|----------------------------------|---|
| <code>self</code> | struct Oracle.Observation[65535] | The stored oracle array |
| <code>time</code> | uint32 | The current block.timestamp |
| <code>secondsAgo</code> | uint32[] | Each amount of time to look back, in seconds, at which point to return an observation |
| <code>tick</code> | int24 | The current tick |
| <code>index</code> | uint16 | The location of a given observation within the oracle array |
| <code>liquidity</code> | uint128 | The current in-range pool liquidity |
| <code>cardinality</code> | uint16 | The number of populated elements in the oracle array |

Return Values:

| Name | Type | Description |
|---|-----------|---|
| <code>tickCumulatives</code> | int56[] | The tick * time elapsed since the pool was first initialized, as of each <code>secondsAgo</code> |
| <code>liquidityCumulatives</code> | uint160[] | The liquidity * time elapsed since the pool was first initialized, as of each <code>secondsAgo</code> |
| Positions represent an owner address' liquidity between a lower and upper tick boundary | | |

Positions store additional state for tracking fees owed to the position

Functions

get

```

function get(
    mapping(bytes32 => struct Position.Info) self,
    address owner,
    int24 tickLower,
    int24 tickUpper
) internal view returns (struct Position.Info position)

```

Returns the Info struct of a position, given an owner and position boundaries

Parameters:

| Name | Type | Description |
|------------------------|--|---|
| <code>self</code> | mapping(bytes32 => struct Position.Info) | The mapping containing all user positions |
| <code>owner</code> | address | The address of the position owner |
| <code>tickLower</code> | int24 | The lower tick boundary of the position |
| <code>tickUpper</code> | int24 | The upper tick boundary of the position |

Return Values:

| Name | Type | Description |
|----------|----------------------|--|
| position | struct Position.Info | The position info struct of the given owners' position |

update

```
function update(
    struct Position.Info self,
    int128 liquidityDelta,
    uint256 feeGrowthInside0X128,
    uint256 feeGrowthInside1X128
) internal
```

Credits accumulated fees to a user's position

Parameters:

| Name | Type | Description |
|---|----------------------|---|
| self | struct Position.Info | The mapping containing all user positions |
| liquidityDelta | int128 | The change in pool liquidity as a result of the position update |
| feeGrowthInside0X128 | uint256 | The all-time fee growth in token0, per unit of liquidity, inside the position's tick boundaries |
| feeGrowthInside1X128 | uint256 | The all-time fee growth in token1, per unit of liquidity, inside the position's tick boundaries |
| Contains methods for safely casting between types | | |

Functions**toUint160**

```
function toUint160(
    uint256 y
) internal pure returns (uint160 z)
```

Cast a uint256 to a uint160, revert on overflow

Parameters:

| Name | Type | Description |
|------|---------|------------------------------|
| y | uint256 | The uint256 to be downcasted |

Return Values:

| Name | Type | Description |
|------|---------|--|
| z | uint160 | The downcasted integer, now type uint160 |

toInt128

```
function toInt128(
    int256 y
) internal pure returns (int128 z)
```

Cast a int256 to a int128, revert on overflow or underflow

Parameters:

| Name | Type | Description |
|------|--------|-----------------------------|
| y | int256 | The int256 to be downcasted |

Return Values:

| Name | Type | Description |
|------|--------|---|
| z | int128 | The downcasted integer, now type int128 |

toInt256

```
function toInt256(
    uint256 y
) internal pure returns (int256 z)
```

Cast a uint256 to a int256, revert on overflow

Parameters:

| Name | Type | Description |
|------|---------|--------------------------|
| y | uint256 | The uint256 to be casted |

Return Values:

| Name | Type | Description |
|--|--------|-------------------------------------|
| z | int256 | The casted integer, now type int256 |
| Contains methods for working with a mapping from tick to 32 bit timestamp values, specifically seconds | | |
| spent outside the tick. | | |

The mapping uses int24 for keys since ticks are represented as int24 and there are 8 (2^3) values per word. Note "seconds outside" is always a relative measurement, only consistent for as long as the lower tick and upper tick have gross liquidity greater than 0.

Functions**initialize**

```
function initialize(
    mapping(int24 => uint256) self,
    int24 tick,
    int24 tickCurrent,
    int24 tickSpacing,
    uint32 time
) internal
```

Called the first time a tick is used to set the seconds outside value. Assumes the tick is not initialized.

Parameters:

| Name | Type | Description |
|-------------|---------------------------|---|
| self | mapping(int24 => uint256) | the packed mapping of tick to seconds outside |
| tick | int24 | the tick to be initialized |
| tickCurrent | int24 | the current tick |
| tickSpacing | int24 | the spacing between usable ticks |
| time | uint32 | the current timestamp |

clear

```
function clear(
    mapping(int24 => uint256) self,
    int24 tick,
    int24 tickSpacing
) internal
```

Called when a tick is no longer used, to clear the seconds outside value of the tick

Parameters:

| Name | Type | Description |
|-------------|---------------------------|---|
| self | mapping(int24 => uint256) | the packed mapping of tick to seconds outside |
| tick | int24 | the tick to be cleared |
| tickSpacing | int24 | the spacing between usable ticks |

cross

```
function cross(
    mapping(int24 => uint256) self,
    int24 tick,
    int24 tickSpacing,
    uint32 time
) internal
```

Called when an initialized tick is crossed to update the seconds outside for that tick. Must be called every time an initialized tick is crossed

Parameters:

| Name | Type | Description |
|-------------|---------------------------|--|
| self | mapping(int24 => uint256) | the packed mapping of tick to seconds outside |
| tick | int24 | the tick to be crossed |
| tickSpacing | int24 | the spacing between usable ticks |
| time | uint32 | the current block timestamp truncated to 32 bits |

get

```
function get(
    mapping(int24 => uint256) self,
    int24 tick,
    int24 tickSpacing
) internal view returns (uint32)
```

Get the seconds outside for an initialized tick. Should be called only on initialized ticks.

Parameters:

| Name | Type | Description |
|-------------|---------------------------|---|
| self | mapping(int24 => uint256) | the packed mapping of tick to seconds outside |
| tick | int24 | the tick to get the seconds outside value for |
| tickSpacing | int24 | the spacing between usable ticks |

Return Values:

| Type | Description |
|--------|-------------------------------------|
| uint32 | seconds outside value for that tick |

secondsInside

```
function secondsInside(
    mapping(int24 => uint256) self,
    int24 tickLower,
    int24 tickUpper,
    int24 tickCurrent,
    int24 tickSpacing
) internal view returns (uint32)
```

Get the seconds inside a tick range, assuming both tickLower and tickUpper are initialized

Parameters:

| Name | Type | Description |
|-------------|---------------------------|--|
| self | mapping(int24 => uint256) | the packed mapping of tick to seconds outside |
| tickLower | int24 | the lower tick for which to get seconds inside |
| tickUpper | int24 | the upper tick for which to get seconds inside |
| tickCurrent | int24 | the current tick |
| tickSpacing | int24 | the spacing between usable ticks |

Return Values:

| Name | Type | Description |
|------|--------|---|
| a | uint32 | relative seconds inside value that can be snapshotted and compared to a later snapshot to compute |

time spent between tickLower and tickUpper, i.e. time that a position's liquidity was in use. Contains the math that uses square root of price as a Q64.96 and liquidity to compute deltas

Functions

getNextSqrtPriceFromAmount0RoundingUp

```
function getNextSqrtPriceFromAmount0RoundingUp(
    uint160 sqrtPX96,
    uint128 liquidity,
    uint256 amount,
    bool add
) internal pure returns (uint160)
```

Gets the next sqrt price given a delta of token0

Always rounds up, because in the exact output case (increasing price) we need to move the price at least far enough to get the desired output amount, and in the exact input case (decreasing price) we need to move the price less in order to not send too much output. The most precise formula for this is $liquidity * sqrtPX96 / (liquidity + amount * sqrtPX96)$, if this is impossible because of overflow, we calculate $liquidity / (liquidity / sqrtPX96 + amount)$.

Parameters:

| Name | Type | Description |
|-----------|---------|---|
| sqrtPX96 | uint160 | The starting price, i.e. before accounting for the token0 delta |
| liquidity | uint128 | The amount of usable liquidity |
| amount | uint256 | How much of token0 to add or remove from virtual reserves |
| add | bool | Whether to add or remove the amount of token0 |

Return Values:

| Type | Description |
|---------|---|
| uint160 | price after adding or removing amount, depending on add |

getNextSqrtPriceFromAmount1RoundingDown

```
function getNextSqrtPriceFromAmount1RoundingDown(
    uint160 sqrtPX96,
    uint128 liquidity,
    uint256 amount,
    bool add
) internal pure returns (uint160)
```

Gets the next sqrt price given a delta of token1

Always rounds down, because in the exact output case (decreasing price) we need to move the price at least far enough to get the desired output amount, and in the exact input case (increasing price) we need to move the price less in order to not send too much output. The formula we compute is within <1 wei of the lossless version: $\sqrt{P}X96 \pm \text{amount} / \text{liquidity}$

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| sqrtPX96 | uint160 | The starting price, i.e., before accounting for the token1 delta |
| liquidity | uint128 | The amount of usable liquidity |
| amount | uint256 | How much of token1 to add, or remove, from virtual reserves |
| add | bool | Whether to add, or remove, the amount of token1 |

Return Values:

| Type | Description |
|---------|---------------------------------------|
| uint160 | price after adding or removing amount |

getNextSqrtPriceFromInput

```
function getNextSqrtPriceFromInput(
    uint160 sqrtPX96,
    uint128 liquidity,
    uint256 amountIn,
    bool zeroForOne
) internal pure returns (uint160 sqrtQX96)
```

Gets the next sqrt price given an input amount of token0 or token1

Throws if price or liquidity are 0, or if the next price is out of bounds

Parameters:

| Name | Type | Description |
|------------|---------|--|
| sqrtPX96 | uint160 | The starting price, i.e., before accounting for the input amount |
| liquidity | uint128 | The amount of usable liquidity |
| amountIn | uint256 | How much of token0, or token1, is being swapped in |
| zeroForOne | bool | Whether the amount in is token0 or token1 |

Return Values:

| Name | Type | Description |
|----------|---------|---|
| sqrtQX96 | uint160 | The price after adding the input amount to token0 or token1 |

getNextSqrtPriceFromOutput

```
function getNextSqrtPriceFromOutput(
    uint160 sqrtPX96,
    uint128 liquidity,
    uint256 amountOut,
    bool zeroForOne
) internal pure returns (uint160 sqrtQX96)
```

Gets the next sqrt price given an output amount of token0 or token1

Throws if price or liquidity are 0 or the next price is out of bounds

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| sqrtPX96 | uint160 | The starting price before accounting for the output amount |
| liquidity | uint128 | The amount of usable liquidity |

| | | |
|------------|---------|---|
| amountOut | uint256 | How much of token0, or token1, is being swapped out |
| zeroForOne | bool | Whether the amount out is token0 or token1 |

Return Values:

| Name | Type | Description |
|----------|---------|--|
| sqrtQX96 | uint160 | The price after removing the output amount of token0 or token1 |

getAmount0Delta

```
function getAmount0Delta(
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    uint128 liquidity,
    bool roundUp
) internal pure returns (uint256 amount0)
```

Gets the amount0 delta between two prices

Calculates $liquidity / \text{sqrt}(lower) - liquidity / \text{sqrt}(upper)$, i.e. $liquidity * (\text{sqrt}(upper) - \text{sqrt}(lower)) / (\text{sqrt}(upper) * \text{sqrt}(lower))$

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioAX96 | uint160 | A sqrt price |
| sqrtRatioBX96 | uint160 | Another sqrt price |
| liquidity | uint128 | The amount of usable liquidity |
| roundUp | bool | Whether to round the amount up or down |

Return Values:

| Name | Type | Description |
|---------|---------|---|
| amount0 | uint256 | Amount of token0 required to cover a position of size liquidity between the two passed prices |

getAmount1Delta

```
function getAmount1Delta(
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    uint128 liquidity,
    bool roundUp
) internal pure returns (uint256 amount1)
```

Gets the amount1 delta between two prices

Calculates $liquidity * (\text{sqrt}(upper) - \text{sqrt}(lower))$

Parameters:

| Name | Type | Description |
|---------------|---------|---|
| sqrtRatioAX96 | uint160 | A sqrt price |
| sqrtRatioBX96 | uint160 | Another sqrt price |
| liquidity | uint128 | The amount of usable liquidity |
| roundUp | bool | Whether to round the amount up, or down |

Return Values:

| Name | Type | Description |
|---------|---------|---|
| amount1 | uint256 | Amount of token1 required to cover a position of size liquidity between the two passed prices |

getAmount0Delta

```
function getAmount0Delta(
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    int128 liquidity
) internal pure returns (int256 amount0)
```

Helper that gets signed token0 delta

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioAX96 | uint160 | A sqrt price |
| sqrtRatioBX96 | uint160 | Another sqrt price |
| liquidity | int128 | The change in liquidity for which to compute the amount0 delta |

Return Values:

| Name | Type | Description |
|---------|--------|--|
| amount0 | int256 | Amount of token0 corresponding to the passed liquidityDelta between the two prices |

getAmount1Delta

```
function getAmount1Delta(
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    int128 liquidity
) internal pure returns (int256 amount1)
```

Helper that gets signed token1 delta

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioAX96 | uint160 | A sqrt price |
| sqrtRatioBX96 | uint160 | Another sqrt price |
| liquidity | int128 | The change in liquidity for which to compute the amount1 delta |

Return Values:

| Name | Type | Description |
|--|--------|--|
| amount1 | int256 | Amount of token1 corresponding to the passed liquidityDelta between the two prices |
| Contains methods for computing the result of a swap within a single tick price range, i.e., a single tick. | | |

Functions

computeSwapStep

```
function computeSwapStep(
    uint160 sqrtRatioCurrentX96,
    uint160 sqrtRatioTargetX96,
    uint128 liquidity,
    int256 amountRemaining,
    uint24 feePips
) internal pure returns (uint160 sqrtRatioNextX96, uint256 amountIn, uint256 amountOut, uint256 feeAmount)
```

Computes the result of swapping some amount in, or amount out, given the parameters of the swap

The fee, plus the amount in, will never exceed the amount remaining if the swap's `amountSpecified` is positive

Parameters:

| Name | Type | Description |
|----------------------------------|----------------------|---|
| <code>sqrtRatioCurrentX96</code> | <code>uint160</code> | The current sqrt price of the pool |
| <code>sqrtRatioTargetX96</code> | <code>uint160</code> | The price that cannot be exceeded, from which the direction of the swap is inferred |
| <code>liquidity</code> | <code>uint128</code> | The usable liquidity |
| <code>amountRemaining</code> | <code>int256</code> | How much input or output amount is remaining to be swapped in/out |
| <code>feePips</code> | <code>uint24</code> | The fee taken from the input amount, expressed in hundredths of a bip |

Return Values:

| Name | Type | Description |
|--|----------------------|---|
| <code>sqrtRatioNextX96</code> | <code>uint160</code> | The price after swapping the amount in/out, not to exceed the price target |
| <code>amountIn</code> | <code>uint256</code> | The amount to be swapped in, of either token0 or token1, based on the direction of the swap |
| <code>amountOut</code> | <code>uint256</code> | The amount to be received, of either token0 or token1, based on the direction of the swap |
| <code>feeAmount</code> | <code>uint256</code> | The amount of input that will be taken as a fee |
| Contains functions for managing tick processes and relevant calculations | | |

Functions

`tickSpacingToMaxLiquidityPerTick`

```
function tickSpacingToMaxLiquidityPerTick(
    int24 tickSpacing
) internal pure returns (uint128)
```

Derives max liquidity per tick from given tick spacing

Executed within the pool constructor

Parameters:

| Name | Type | Description |
|--------------------------|--------------------|---|
| <code>tickSpacing</code> | <code>int24</code> | The amount of required tick separation, realized in multiples of <code>tickSpacing</code> e.g., a <code>tickSpacing</code> of 3 requires ticks to be initialized every 3rd tick i.e., ..., -6, -3, 0, 3, 6, ... |

Return Values:

| Type | Description |
|----------------------|------------------------|
| <code>uint128</code> | max liquidity per tick |

`getFeeGrowthInside`

```
function getFeeGrowthInside(
    mapping(int24 => struct Tick.Info) self,
    int24 tickLower,
    int24 tickUpper,
    int24 tickCurrent,
    uint256 feeGrowthGlobal0X128,
    uint256 feeGrowthGlobal1X128
) internal view returns (uint256 feeGrowthInside0X128, uint256 feeGrowthInside1X128)
```

Retrieves fee growth data

Parameters:

| Name | Type | Description |
|----------------------|------------------------------------|---|
| self | mapping(int24 => struct Tick.Info) | The mapping containing all tick information for initialized ticks |
| tickLower | int24 | The lower tick boundary of the position |
| tickUpper | int24 | The upper tick boundary of the position |
| tickCurrent | int24 | The current tick |
| feeGrowthGlobal0X128 | uint256 | The all-time global fee growth, per unit of liquidity, in token0 |
| feeGrowthGlobal1X128 | uint256 | The all-time global fee growth, per unit of liquidity, in token1 |

Return Values:

| Name | Type | Description |
|----------------------|---------|---|
| feeGrowthInside0X128 | uint256 | The all-time fee growth in token0, per unit of liquidity, inside the position's tick boundaries |
| feeGrowthInside1X128 | uint256 | The all-time fee growth in token1, per unit of liquidity, inside the position's tick boundaries |

update

```
function update(
    mapping(int24 => struct Tick.Info) self,
    int24 tick,
    int24 tickCurrent,
    int128 liquidityDelta,
    uint256 feeGrowthGlobal0X128,
    uint256 feeGrowthGlobal1X128,
    bool upper,
    uint128 maxLiquidity
) internal returns (bool flipped)
```

Updates a tick and returns true if the tick was flipped from initialized to uninitialized, or vice versa

Parameters:

| Name | Type | Description |
|----------------------|------------------------------------|--|
| self | mapping(int24 => struct Tick.Info) | The mapping containing all tick information for initialized ticks |
| tick | int24 | The tick that will be updated |
| tickCurrent | int24 | The current tick |
| liquidityDelta | int128 | A new amount of liquidity to be added (subtracted) when tick is crossed from left to right (right to left) |
| feeGrowthGlobal0X128 | uint256 | The all-time global fee growth, per unit of liquidity, in token0 |
| feeGrowthGlobal1X128 | uint256 | The all-time global fee growth, per unit of liquidity, in token1 |
| upper | bool | true for updating a position's upper tick, or false for updating a position's lower tick |
| maxLiquidity | uint128 | The maximum liquidity allocation for a single tick |

Return Values:

| Name | Type | Description |
|---------|------|---|
| flipped | bool | Whether the tick was flipped from initialized to uninitialized, or vice versa |

clear

```
function clear(
    mapping(int24 => struct Tick.Info) self,
```

```

    int24 tick
) internal

```

Clears tick data

Parameters:

| Name | Type | Description |
|------|------------------------------------|---|
| self | mapping(int24 => struct Tick.Info) | The mapping containing all initialized tick information for initialized ticks |
| tick | int24 | The tick that will be cleared |

cross

```

function cross(
    mapping(int24 => struct Tick.Info) self,
    int24 tick,
    uint256 feeGrowthGlobal0X128,
    uint256 feeGrowthGlobal1X128
) internal returns (int128 liquidityNet)

```

Transitions to next tick as needed by price movement

Parameters:

| Name | Type | Description |
|----------------------|------------------------------------|---|
| self | mapping(int24 => struct Tick.Info) | The mapping containing all tick information for initialized ticks |
| tick | int24 | The destination tick of the transition |
| feeGrowthGlobal0X128 | uint256 | The all-time global fee growth, per unit of liquidity, in token0 |
| feeGrowthGlobal1X128 | uint256 | The all-time global fee growth, per unit of liquidity, in token1 |

Return Values:

| Name | Type | Description |
|--|--------|--|
| liquidityNet | int128 | The amount of liquidity added (subtracted) when tick is crossed from left to right (right to left) |
| Stores a packed mapping of tick index to its initialized state | | |

The mapping uses int16 for keys since ticks are represented as int24 and there are 256 (2^8) values per word.

Functions

flipTick

```

function flipTick(
    mapping(int16 => uint256) self,
    int24 tick,
    int24 tickSpacing
) internal

```

Flips the initialized state for a given tick from false to true, or vice versa

Parameters:

| Name | Type | Description |
|-------------|---------------------------|---------------------------------------|
| self | mapping(int16 => uint256) | The mapping in which to flip the tick |
| tick | int24 | The tick to flip |
| tickSpacing | int24 | The spacing between usable ticks |

nextInitializedTickWithinOneWord

```

function nextInitializedTickWithinOneWord(
    mapping(int16 => uint256) self,
    int24 tick,
    int24 tickSpacing,
    bool lte
) internal view returns (int24 next, bool initialized)

```

Returns the next initialized tick contained in the same word (or adjacent word) as the tick that is either to the left (less than or equal to) or right (greater than) of the given tick

Parameters:

| Name | Type | Description |
|-------------|---------------------------|---|
| self | mapping(int16 => uint256) | The mapping in which to compute the next initialized tick |
| tick | int24 | The starting tick |
| tickSpacing | int24 | The spacing between usable ticks |
| lte | bool | Whether to search for the next initialized tick to the left (less than or equal to the starting tick) |

Return Values:

| Name | Type | Description |
|---|-------|--|
| next | int24 | The next initialized or uninitialized tick up to 256 ticks away from the current tick |
| initialized | bool | Whether the next tick is initialized, as the function only searches within up to 256 ticks |
| Computes sqrt price for ticks of size 1.0001, i.e. $\sqrt{1.0001^{tick}}$ as fixed point Q64.96 numbers. Supports prices between 2^{-128} and 2^{128} | | |
| | | |

Functions

getSqrtRatioAtTick

```

function getSqrtRatioAtTick(
    int24 tick
) internal pure returns (uint160 sqrtPriceX96)

```

Calculates $\sqrt{1.0001^{tick}} * 2^{96}$

Throws if $|tick| > \text{max tick}$

Parameters:

| Name | Type | Description |
|------|-------|--------------------------------------|
| tick | int24 | The input tick for the above formula |

Return Values:

| Name | Type | Description |
|--------------|---------|--|
| sqrtPriceX96 | uint160 | A Fixed point Q64.96 number representing the sqrt of the ratio of the two assets (token1/token0) |

at the given tick

getTickAtSqrtRatio

```

function getTickAtSqrtRatio(
    uint160 sqrtPriceX96
) internal pure returns (int24 tick)

```

Calculates the greatest tick value such that $\text{getRatioAtTick}(tick) \leq \text{ratio}$

Throws in case sqrtPriceX96 < MIN_SQRT_RATIO, as MIN_SQRT_RATIO is the lowest value getRatioAtTick may ever return.

Parameters:

| Name | Type | Description |
|--------------|---------|--|
| sqrtPriceX96 | uint160 | The sqrt ratio for which to compute the tick as a Q64.96 |

Return Values:

| Name | Type | Description |
|--|-------|--|
| tick | int24 | The greatest tick for which the ratio is less than or equal to the input ratio |
| Contains helper methods for interacting with ERC20 tokens that do not consistently return true/false | | |

Functions

safeTransfer

```
function safeTransfer(
    address token,
    address to,
    uint256 value
) internal
```

Transfers tokens from msg.sender to a recipient

Calls transfer on token contract, errors with TF if transfer fails

Parameters:

| Name | Type | Description |
|--|---------|---|
| token | address | The contract address of the token which will be transferred |
| to | address | The recipient of the transfer |
| value | uint256 | The value of the transfer |
| Contains methods that perform common math functions but do not do any overflow or underflow checks | | |

Functions

divRoundingUp

```
function divRoundingUp(
    uint256 x,
    uint256 y
) internal pure returns (uint256 z)
```

Returns ceil(x / y)

panics if y == 0

Parameters:

| Name | Type | Description |
|------|---------|--------------|
| x | uint256 | The dividend |
| y | uint256 | The divisor |

Return Values:

| Name | Type | Description |
|------|---------|---------------------------|
| z | uint256 | The quotient, ceil(x / y) |

id: error-codes title: Error Codes

LiquidityMath.sol

- `LS` : Liquidity Sub
- `LA` : Liquidity Add

Oracle.sol

- `OLD` : The target must be chronologically after the oldest observation
- `I` : The pool has not been initialized

Position.sol

- `NP` : Burn cannot be called for a position with 0 liquidity

Tick.sol

- `LO` : LiquidityGrossAfter must be less than MaxLiquidity

TickMath.sol

- `T` : The given tick must be less than, or equal to, the maximum tick
- `R` : second inequality must be < because the price can never reach the price at the max tick

TransferHelper.sol

- `TF` : Transfer Failed : errors with TF if transfer fails

UniswapV3Pool.sol

- `LOK` : The reentrancy guard. A transaction cannot re-enter the pool mid-swap
- `TLU` : The lower tick must be below the upper tick
- `TLM` : The lower tick must be greater, or equal to, the minimum tick
- `TUM` : The upper tick must be lesser than, or equal to, the maximum tick
- `AI` : The pool is already initialized
- `M0` : Mint 0, The balance of token0 in the given pool before minting must be less than, or equal to, the balance after minting
- `M1` : Mint 1, The balance of token1 in the given pool before minting must be less than, or equal to, the balance after minting
- `AS` : `amountSpecified` cannot be zero
- `SPL` : Square root price limit
- `IIA` : Insufficient input amount, an insufficient amount of input token was sent during the callback
- `L` : Liquidity in the pool must be greater than zero for a flash to be executed
- `F0` : The balance of token0 in the given pool before the flash transaction must be less than, or equal to, the balance of token0 after the flash plus the fee
- `F1` : The balance of token1 in the given pool before the flash transaction must be less than, or equal to, the balance of token1 after the flash plus the fee

id: overview title: Overview

The updated reference for the newly deployed Governor Bravo is available via [Etherscan](#), some of the reference material below may be out of date.

The Uniswap protocol is governed and upgraded by UNI token holders, using three distinct components; the UNI token, governance module, and Timelock. Together, these contracts allow the community to propose, vote, and implement changes to the uniswap protocol.

Any addresses with more than 2.5M UNI (0.25% of total supply) delegated to it may propose governance actions, which contain finished, executable code. When a proposal is created, the community can cast their votes during a 7 day voting period. If there are more 'For' votes than 'Against' (i.e. a simple majority), and the number of 'For' votes >40M (meeting the quorum), it is queued in the Timelock, and may be executed in a minimum of 2 days.

Timelock

The Timelock contract can modify system parameters, logic, and contracts in a 'time-delayed, opt-out' upgrade pattern. Timelock has a hard-coded minimum delay of 2 days, which is the least amount of notice possible for a governance action. Each proposed action will be published at a minimum of 2 days in the future from the time of announcement. Major upgrades, such as changing the risk system, may have up to a 30 day delay. Timelock is controlled by the governance module; pending and completed governance actions can be monitored on the Timelock Dashboard.

Key Events

DelegateChanged

```
DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate)
```

Emitted when an account changes its delegate.

DelegateVotesChanged

```
DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance)
```

Emitted when a delegate account's vote balance changes.

ProposalCreated

```
ProposalCreated(uint id, address proposer, address[] targets, uint[] values, string[] signatures, bytes[] calldatas, uint startBlock, uint endBlock, string description)
```

Emitted when a new proposal is created.

VoteCast

```
VoteCast(address voter, uint proposalId, bool support, uint votes)
```

Emitted when a vote has been cast on a proposal.

ProposalCanceled

```
ProposalCanceled(uint id)
```

Emitted when a proposal has been canceled.

ProposalQueued

```
ProposalQueued(uint id, uint eta)
```

Emitted when a proposal has been queued in the Timelock.

ProposalExecuted

```
ProposalExecuted(uint id)
```

Emitted when a proposal has been executed in the Timelock.

Read-Only Functions: UNI

Get Current Votes

```
function getCurrentVotes(address account) returns (uint96)
```

Returns the balance of votes for an account as of the current block.

| Name | Type | |
|---------|---------|--|
| account | address | Address of the account of which to retrieve the number of votes. |

Get Prior Votes

```
function getPriorVotes(address account, uint blockNumber) returns (uint96)
```

Returns the prior number of votes for an account at a specific block number. The block number passed must be a finalized block or the function will revert.

| Name | Type | |
|-------------|---------|--|
| account | address | Address of the account of which to retrieve the prior number of votes. |
| blocknumber | uint | The block number at which to retrieve the prior number of votes. |
| unnamed | uint96 | The number of prior votes |

State-Changing Functions: UNI

Delegate

```
function delegate(address delegatee)
```

Delegate votes from the sender to the delegatee. Users can delegate to 1 address at a time, and the number of votes added to the delegatee's vote count is equivalent to the balance of UNI in the user's account. Votes are delegated from the current block and onward, until the sender delegates again, or transfers their UNI.

| Name | Type | |
|-----------|---------|--|
| delegatee | address | The address to which msg.sender wishes to delegate their votes to. |

Delegate By Signature

```
function delegateBySig(address delegatee, uint nonce, uint expiry, uint8 v, bytes32 r, bytes32 s)
```

Delegate votes from the sender to the delegatee. Users can delegate to 1 address at a time, and the number of votes added to the delegatee's vote count is equivalent to the balance of UNI in the user's account. Votes are delegated from the current block and onward, until the sender delegates again, or transfers their UNI.

| Name | Type | |
|-----------|---------|---|
| delegatee | address | The address to which msg.sender wishis to delegate their vote to |
| nonce | uint | The contract state required to match the signature. This can be retrieved from the contract's public nonces mapping |
| expiry | uint | The time when the signature expires. A block timestamp in seconds since the unix epoch. |
| v | uint | The recovery byte of the signature. |
| r | bytes32 | Half of the ECDSA signature pair. |
| s | bytes32 | Half of the ECDSA signature pair. |

Read-Only Functions: Governor Alpha

Quorum Votes

```
function quorumVotes() public pure returns (uint)
```

Returns the minimum number of votes required for a proposal to succeed.

Proposal Threshold

```
function proposalThreshold() returns (uint)
```

Returns the minimum number of votes required for an account to create a proposal.

Proposal Max Operations

```
function proposalMaxOperations() returns (uint)
```

Returns the maximum number of actions that can be included in a proposal. Actions are functions calls that will be made when a proposal succeeds and executes.

Voting Delay

```
function votingDelay() returns (uint)
```

Returns the number of blocks to wait before voting on a proposal may begin. This value is added to the current block number when a proposal is created.

Voting Period

```
function votingPeriod() returns (uint)
```

Returns the duration of voting on a proposal, in blocks.

Get Actions

```
function getActions(uint proposalId) returns (uint proposalId) public view returns (address[] memory targets,  
uint[] memory values, string[] memory signatures, bytes[] memory calldatas)
```

Gets the actions of a selected proposal. Pass a proposal ID and get the targets, values, signatures and calldatas of that proposal.

| Name | Type | |
|------------|------|--------------------|
| proposalId | uint | ID of the proposal |

Returns:

- Array of addresses of contracts the proposal calls.
- Array of unsigned integers the proposal uses as values.
- Array of strings of the proposal's signatures.
- Array of calldata bytes of the proposal.

Get Receipt

```
function getReceipt(uint proposalId, address voter) returns (Receipt memory)
```

Returns a proposal ballot receipt of a given voter.

| Name | Type | |
|------|------|--|
| | | |

| | | |
|------------|---------|--|
| proposalId | uint | ID of the proposal in which to get a voter's ballot receipt. |
| voter | address | Address of the account of a proposal voter. |
| Receipt | struct | A Receipt struct for the ballot of the voter address. |

State

```
function state(uint proposalId) returns (ProposalState)
```

Returns enum of type ProposalState, possible types are:

- Pending
- Active
- Canceled
- Defeated
- Succeeded
- Queued
- Expired
- Executed

| Name | Type | |
|------------|------|--------------------|
| proposalId | uint | ID of the proposal |

State-Changing Functions: Governor Alpha

Propose

```
function propose(address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldatas, string memory description) returns (uint)
```

Creates a Proposal to change the protocol.

Proposals will be voted on by delegated voters. If there is sufficient support before the voting period ends, the proposal shall be automatically enacted. Enacted proposals are queued and executed in the Timelock contract.

The sender must hold more UNI than the current proposal threshold (proposalThreshold()) as of the immediately previous block. The proposal can have up to 10 actions (based on proposalMaxOperations()).

The proposer cannot create another proposal if they currently have a pending or active proposal. It is not possible to queue two identical actions in the same block (due to a restriction in the Timelock), therefore actions in a single proposal must be unique, and unique proposals that share an identical action must be queued in different blocks.

| Name | Type | |
|-------------|---------|---|
| targets | address | The ordered list of target addresses for calls to be made during proposal execution. This array must be the same length as all other array parameters in this function. |
| values | uint | The ordered list of values (i.e. msg.value) to be passed to the calls made during proposal execution. This array must be the same length as all other array parameters in this function |
| signatures | string | The ordered list of function signatures to be passed during execution. This array must be the same length as all other array parameters in this function. |
| calldatas | bytes | The ordered list of data to be passed to each individual function call during proposal execution. This array must be the same length as all other array parameters in this function. |
| description | string | A human readable description of the proposal and the changes it will enact. |
| Unnamed | uint | Returns ID of the new proposal |

Queue

```
function queue(uint proposalId)
```

After a proposal has succeeded, any address can call the queue method to move the proposal into the Timelock queue. A proposal can only be queued if it has succeeded.

| Name | Type | |
|------------|------|-----------------------------------|
| proposalId | uint | ID of a given successful proposal |

Execute

```
function execute(uint proposalId) payable
```

After the Timelock delay period, any account may invoke the execute method to apply the changes from the proposal to the target contracts. This will invoke each of the actions described in the proposal. This function is payable so the Timelock contract can invoke payable functions that were selected in the proposal.

| Name | Type | |
|------------|------|-----------------------------------|
| proposalId | uint | ID of a given successful proposal |

Cancel

```
function cancel(uint proposalId)
```

Cancel a proposal that has not yet been executed. The Guardian is the only one who may execute this unless the proposer does not maintain the delegates required to create a proposal. If the proposer does not have more delegates than the proposal threshold, anyone can cancel the proposal.

| Name | Type | |
|------------|------|----------------------------|
| proposalId | uint | ID of a proposal to cancel |

Cast Vote

```
function castVote(uint proposalId, bool support)
```

Cast a vote on a proposal. The account's voting weight is determined by its number of delegated votes at the time the proposal becomes active.

| Name | Type | |
|------------|------|---|
| proposalId | uint | ID of a given successful proposal |
| support | bool | A boolean of true for 'yes' or false for 'no' on the proposal vote. |

Cast Vote By Signature

```
function castVoteBySig(uint proposalId, bool support, uint8 v, bytes32 r, bytes32 s)
```

Cast a vote on a proposal. The account's voting weight is determined by its number of delegated votes at the time the proposal became active. This method has the same purpose as Cast Vote, but instead enables offline signatures to participate in governance voting. For more details on how to create an offline signature, review EIP-712.

| Name | Type | |
|------------|---------|---|
| proposalId | uint | ID of a given successful proposal |
| support | bool | A boolean of true for 'yes' or false for 'no' on the proposal vote. |
| expiry | uint | The time when the signature expires. A block timestamp in seconds since the unix epoch. |
| v | uint | The recovery byte of the signature. |
| r | bytes32 | Half of the ECDSA signature pair. |
| s | bytes32 | Half of the ECDSA signature pair. |

id: overview title: Overview sidebar_position: 1

Uniswap V3 is a binary smart contract system comprised of many libraries, which together make the Core and Periphery.

Core contracts provide fundamental safety guarantees for all parties interacting with Uniswap. They define the logic of pool generation, the pools themselves, and the interactions involving the respective assets therein.

Periphery contracts interact with one or more Core contracts but are not part of the core. They are designed to provide methods of interacting with the core that increase clarity and user safety.

External calls will primarily call the periphery interfaces. Externally available functions are all viewable in the reference documentation. Internal functions are viewable on the Uniswap V3 Github repo.

Core

[Core Source Code](#)

The core consists of a single factory, a pool deployer, and the many pools the factory will create.

A significant amount of care and attention has been given to gas optimization in the core contracts. The result is a substantial reduction in gas costs for all protocol interactions compared to V2, at the cost of a reduction in code clarity.

Factory

[Factory Reference](#)

The factory defines the logic for generating pools. A pool is defined by two tokens, which make up the asset pair, and a fee. There can be multiple pools of the same asset pair, distinguished only by their swap fee.

Pools

[Pool Reference](#)

Pools primarily serve as automated market makers for the paired assets. Additionally, they expose price oracle data and may be used as an asset source for flash transactions.

Periphery

The periphery is a constellation of smart contracts designed to support domain-specific interactions with the core. As the Uniswap protocol is a permissionless system, the contracts described below have no special privileges and are only a small subset of possible periphery-like contracts.

SwapRouter

[Swap Router Reference](#)

[Swap Router Interface](#)

The swap router supports all the basic requirements of a front-end offering trading. It natively supports single trades (x to y) and multihop trades (e.g. x to y to z).

Nonfungible Position Manager

[Nonfungible Position Manager Reference](#)

[Nonfungible Position Manager Interface](#)

The position manager handles the logic transactions involving the creation, adjustment, or exiting of positions.

Oracle

[Oracle Reference](#)

The oracle provides price and liquidity data useful for a wide variety of system designs, and is available in every deployed pool.

Periphery Libraries

[Periphery Libraries](#)

The libraries provide a variety of helper functions developers may need, like calculating pool addresses, safe transfer functions, and more. Wraps Uniswap V3 positions in the ERC721 non-fungible token interface

Functions

constructor

```
function constructor()
) public
```

positions

```
function positions(
    uint256 tokenId
) external view returns (uint96 nonce, address operator, address token0, address token1, uint24 fee, int24
    tickLower, int24 tickUpper, uint128 liquidity, uint256 feeGrowthInside0LastX128, uint256 feeGrowthInside1LastX128,
    uint128 tokensOwed0, uint128 tokensOwed1)
```

Returns the position information associated with a given token ID.

Throws if the token ID is not valid.

Parameters:

| Name | Type | Description |
|---------|---------|--|
| tokenId | uint256 | The ID of the token that represents the position |

Return Values:

| Name | Type | Description |
|--------------------------|---------|--|
| nonce | uint96 | The nonce for permits |
| operator | address | The address that is approved for spending |
| token0 | address | The address of the token0 for a specific pool |
| token1 | address | The address of the token1 for a specific pool |
| fee | uint24 | The fee associated with the pool |
| tickLower | int24 | The lower end of the tick range for the position |
| tickUpper | int24 | The higher end of the tick range for the position |
| liquidity | uint128 | The liquidity of the position |
| feeGrowthInside0LastX128 | uint256 | The fee growth of token0 as of the last action on the individual position |
| feeGrowthInside1LastX128 | uint256 | The fee growth of token1 as of the last action on the individual position |
| tokensOwed0 | uint128 | The uncollected amount of token0 owed to the position as of the last computation |
| tokensOwed1 | uint128 | The uncollected amount of token1 owed to the position as of the last computation |

mint

```
function mint(
    struct INonfungiblePositionManager.MintParams params
) external returns (uint256 tokenId, uint128 liquidity, uint256 amount0, uint256 amount1)
```

Creates a new position wrapped in a NFT

Call this when the pool does exist and is initialized. Note that if the pool is created but not initialized a method does not exist, i.e. the pool is assumed to be initialized.

Parameters:

| Name | Type | Description |
|--------|---|---|
| params | struct INonfungiblePositionManager.MintParams | The params necessary to mint a position, encoded as <code>MintParams</code> in calldata |

Return Values:

| Name | Type | Description |
|---------|---------|---|
| tokenId | uint256 | The ID of the token that represents the minted position |

| | | |
|-----------|---------|---|
| liquidity | uint128 | The amount of liquidity for this position |
| amount0 | uint256 | The amount of token0 |
| amount1 | uint256 | The amount of token1 |

tokenURI

```
function tokenURI(
    uint256 tokenId
) public view returns (string)
```

Returns a URI describing a particular token ID

Parameters:

| Name | Type | Description |
|---------|---------|---|
| tokenId | uint256 | The ID of the token that represents the minted position |

Return Values:

A base64 string with the URI data.

baseURI

```
function baseURI(
) public returns (string)
```

increaseLiquidity

```
function increaseLiquidity(
    struct INonfungiblePositionManager.IncreaseLiquidityParams params
) external returns (uint128 liquidity, uint256 amount0, uint256 amount1)
```

Increases the amount of liquidity in a position, with tokens paid by the `msg.sender`

Parameters:

| Name | Type | Description |
|--------|--|---|
| params | struct INonfungiblePositionManager.IncreaseLiquidityParams | tokenId The ID of the token for which liquidity is being increased, |

Return Values:

| Name | Type | Description |
|-----------|---------|--|
| liquidity | uint128 | The new liquidity amount as a result of the increase |
| amount0 | uint256 | The amount of token0 to achieve resulting liquidity |
| amount1 | uint256 | The amount of token1 to achieve resulting liquidity |

decreaseLiquidity

```
function decreaseLiquidity(
    struct INonfungiblePositionManager.DecreaseLiquidityParams params
) external returns (uint256 amount0, uint256 amount1)
```

Decreases the amount of liquidity in a position and accounts it to the position

Parameters:

| Name | Type | Description |
|--------|--|---|
| params | struct INonfungiblePositionManager.DecreaseLiquidityParams | tokenId The ID of the token for which liquidity is being decreased, |

Return Values:

| Name | Type | Description |
|---------|---------|--|
| amount0 | uint256 | The amount of token0 accounted to the position's tokens owed |
| amount1 | uint256 | The amount of token1 accounted to the position's tokens owed |

collect

```
function collect(
    struct INonfungiblePositionManager.CollectParams params
) external returns (uint256 amount0, uint256 amount1)
```

Collects up to a maximum amount of fees owed to a specific position to the recipient

Parameters:

| Name | Type | Description |
|--------|--|---|
| params | struct INonfungiblePositionManager.CollectParams | tokenId The ID of the NFT for which tokens are being collected, |

recipient The account that should receive the tokens, amount0Max The maximum amount of token0 to collect, amount1Max The maximum amount of token1 to collect

Return Values:

| Name | Type | Description |
|---------|---------|--|
| amount0 | uint256 | The amount of fees collected in token0 |
| amount1 | uint256 | The amount of fees collected in token1 |

burn

```
function burn(
    uint256 tokenId
) external
```

Burns a token ID, which deletes it from the NFT contract. The token must have 0 liquidity and all tokens must be collected first.

Parameters:

| Name | Type | Description |
|---------|---------|--|
| tokenId | uint256 | The ID of the token that is being burned |

_getAndIncrementNonce

```
function _getAndIncrementNonce(
) internal returns (uint256)
```

getApproved

```
function getApproved(
) public view returns (address)
```

Returns the account approved for `tokenId` token. Requirements:

- `tokenId` must exist.

_approve

```
function _approve(
) internal
```

Overrides `_approve` to use the operator in the position, which is packed with the position permit nonce Produces a string containing the data URI for a JSON metadata string

Functions

constructor

```
function constructor()
) public
```

tokenURI

```
function tokenURI(
contract INonfungiblePositionManager positionManager,
uint256 tokenId
) external returns (string)
```

Produces the URI describing a particular token ID for a position manager

Note this URI may be a data: URI with the JSON contents directly inlined

Parameters:

| Name | Type | Description |
|-----------------|--------------------------------------|--|
| positionManager | contract INonfungiblePositionManager | The position manager for which to describe the token |
| tokenId | uint256 | The ID of the token for which to produce a description, which may not be valid |

Return Values:

| Name | Type | Description |
|------|--------------------------------------|--------------------------------------|
| The | contract INonfungiblePositionManager | URI of the ERC721-compliant metadata |

flipRatio

```
function flipRatio(
) public returns (bool)
```

tokenRatioPriority

```
function tokenRatioPriority(
) public returns (int256)
```

Router for stateless execution of swaps against Uniswap V3

| Input parameters are viewable on the [Swap Router Interface](#)

Functions

constructor

```
function constructor()
) public
```

uniswapV3SwapCallback

```
function uniswapV3SwapCallback(
int256 amount0Delta,
int256 amount1Delta,
bytes data
) external
```

Called to `msg.sender` after executing a swap via `IUniswapV3Pool#swap`.

In the implementation you must pay the pool tokens owed for the swap. The caller of this method must be checked to be a `UniswapV3Pool` deployed by the canonical `UniswapV3Factory`. `amount0Delta` and `amount1Delta` can both be 0 if no tokens were swapped.

Parameters:

| Name | Type | Description |
|--------------|--------|---|
| amount0Delta | int256 | The amount of token0 that was sent (negative) or must be received (positive) by the pool by the end of the swap. If positive, the callback must send that amount of token0 to the pool. |
| amount1Delta | int256 | The amount of token1 that was sent (negative) or must be received (positive) by the pool by the end of the swap. If positive, the callback must send that amount of token1 to the pool. |
| data | bytes | Any data passed through by the caller via the IUniswapV3PoolActions#swap call |

exactInputSingle

```
function exactInputSingle(
    struct ISwapRouter.ExactInputSingleParams params
) external returns (uint256 amountOut)
```

Swaps `amountIn` of one token for as much as possible of another token

Parameters:

| Name | Type | Description |
|--------|---|---|
| params | struct ISwapRouter.ExactInputSingleParams | The parameters necessary for the swap, encoded as <code>ExactInputSingleParams</code> in calldata |

Return Values:

| Name | Type | Description |
|-----------|---------|----------------------------------|
| amountOut | uint256 | The amount of the received token |

exactInput

```
function exactInput(
    struct ISwapRouter.ExactInputParams params
) external returns (uint256 amountOut)
```

Swaps `amountIn` of one token for as much as possible of another along the specified path

Parameters:

| Name | Type | Description |
|--------|-------------------------------------|---|
| params | struct ISwapRouter.ExactInputParams | The parameters necessary for the multi-hop swap, encoded as <code>ExactInputParams</code> in calldata |

Return Values:

| Name | Type | Description |
|-----------|---------|----------------------------------|
| amountOut | uint256 | The amount of the received token |

exactOutputSingle

```
function exactOutputSingle(
    struct ISwapRouter.ExactOutputSingleParams params
) external returns (uint256 amountIn)
```

Swaps as little as possible of one token for `amountOut` of another token

Parameters:

| Name | Type | Description |
|--------|--|--|
| params | struct ISwapRouter.ExactOutputSingleParams | The parameters necessary for the swap, encoded as <code>ExactOutputSingleParams</code> in calldata |

Return Values:

| Name | Type | Description |
|----------|---------|-------------------------------|
| amountIn | uint256 | The amount of the input token |

exactOutput

```
function exactOutput(
    struct ISwapRouter.ExactOutputParams params
) external returns (uint256 amountIn)
```

Swaps as little as possible of one token for `amountOut` of another along the specified path (reversed)

Parameters:

| Name | Type | Description |
|--------|--------------------------------------|--|
| params | struct ISwapRouter.ExactOutputParams | The parameters necessary for the multi-hop swap, encoded as <code>ExactOutputParams</code> in calldata |

Return Values:

| Name | Type | Description |
|----------|---------|-------------------------------|
| amountIn | uint256 | The amount of the input token |

Functions**constructor**

```
function constructor()
) public
```

receive

```
function receive(
) external
```

migrate

```
function migrate(
) external
```

Base contract that is overridden for tests

Functions**_blockTimestamp**

```
function _blockTimestamp(
) internal view returns (uint256)
```

Method that exists purely to be overridden for tests

Return Values:

| Type | Description |
|---|-------------------------|
| uint256 | current block timestamp |
| Nonfungible tokens that support an approve via signature, i.e. permit | |

Functions

_getAndIncrementNonce

```
function _getAndIncrementNonce(
    uint256 tokenId
) internal virtual returns (uint256)
```

Gets the current nonce for a token ID and then increments it, returning the original value

constructor

```
function constructor(
    string memory name_,
    string memory symbol_,
    string memory version_
) internal
```

Computes the nameHash and versionHash

DOMAIN_SEPARATOR

```
function DOMAIN_SEPARATOR(
) public view override returns (bytes32)
```

The domain separator used in the permit signature

Return Values:

| Type | Description |
|---------|---|
| bytes32 | domain separator used in encoding of permit signature |

permit

```
function permit(
    address spender,
    uint256 tokenId,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external
```

Approve of a specific token ID for spending by spender via signature

Parameters:

| Name | Type | Description |
|--|---------|--|
| spender | address | The account that is being approved |
| tokenId | uint256 | The ID of the token that is being approved for spending |
| deadline | uint256 | The deadline timestamp by which the call must be mined for the approve to work |
| v | uint8 | Must produce valid secp256k1 signature from the holder along with r and s |
| r | bytes32 | Must produce valid secp256k1 signature from the holder along with v and s |
| s | bytes32 | Must produce valid secp256k1 signature from the holder along with r and v |
| Internal functions for safely managing liquidity in Uniswap V3 | | |

Parameter Structs

AddLiquidityParams

```
struct AddLiquidityParams {
    address token0;
    address token1;
    uint24 fee;
    address recipient;
    int24 tickLower;
    int24 tickUpper;
    uint256 amount0Desired;
    uint256 amount1Desired;
    uint256 amount0Min;
    uint256 amount1Min;
}
```

Functions

uniswapV3MintCallback

```
function uniswapV3MintCallback(
    uint256 amount0Owed,
    uint256 amount1Owed,
    bytes data
) external
```

Called to `msg.sender` after minting liquidity to a position from `IUniswapV3Pool#mint`.

In the implementation you must pay the pool tokens owed for the minted liquidity. The caller of this method must be checked to be a `UniswapV3Pool` deployed by the canonical `UniswapV3Factory`.

Parameters:

| Name | Type | Description |
|-------------|---------|--|
| amount0Owed | uint256 | The amount of token0 due to the pool for the minted liquidity |
| amount1Owed | uint256 | The amount of token1 due to the pool for the minted liquidity |
| data | bytes | Any data passed through by the caller via the <code>IUniswapV3PoolActions#mint</code> call |

addLiquidity

```
function addLiquidity(
    AddLiquidityParams memory params
) internal returns (uint128 liquidity, uint256 amount0, uint256 amount1, contract IUniswapV3Pool pool)
```

Add liquidity to an initialized pool Enables calling multiple methods in a single call to the contract

Functions

multicall

```
function multicall(
    bytes[] data
) external payable override returns (bytes[] results)
```

Call multiple functions in the current contract and return the data from all of them if they all succeed

The `msg.value` should not be trusted for any method callable from `multicall`.

Parameters:

| Name | Type | Description |
|------|---------|--|
| data | bytes[] | The encoded function data for each of the calls to make to this contract |

Return Values:

| Name | Type | Description |
|---|---------|---|
| results | bytes[] | The results from each of the calls passed in via data |
| Immutable state used by periphery contracts | | |

Functions

constructor

```
function constructor(
    address _factory, address _WETH9
) internal
```

Functions

receive

```
function receive(
) external
```

unwrapWETH9

```
function unwrapWETH9(
    uint256 amountMinimum,
    address recipient
) external
```

Unwraps the contract's WETH9 balance and sends it to recipient as ETH.

The amountMinimum parameter prevents malicious contracts from stealing WETH9 from users.

Parameters:

| Name | Type | Description |
|---------------|---------|---------------------------------------|
| amountMinimum | uint256 | The minimum amount of WETH9 to unwrap |
| recipient | address | The address receiving ETH |

sweepToken

```
function sweepToken(
    address token,
    uint256 amountMinimum,
    address recipient
) external
```

Transfers the full amount of a token held by this contract to recipient

The amountMinimum parameter prevents malicious contracts from stealing the token from users

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| token | address | The contract address of the token which will be transferred to recipient |
| amountMinimum | uint256 | The minimum amount of token required for a transfer |
| recipient | address | The destination address of the token |

refundETH

```
function refundETH(
) external
```

Refunds any ETH balance held by this contract to the `msg.sender`

Useful for bundling with mint or increase liquidity that uses ether, or exact output swaps that use ether for the input amount

pay

```
function pay(
    address token,
    address payer,
    address recipient,
    uint256 value
) internal
```

Parameters:

| Name | Type | Description |
|-----------|---------|--------------------------------------|
| token | address | The token to pay |
| payer | address | The entity that must pay |
| recipient | address | The entity that will receive payment |
| value | uint256 | The amount to pay |

Functions

unwrapWETH9WithFee

```
function unwrapWETH9WithFee(
    uint256 amountMinimum,
    address recipient,
    uint256 feeBips,
    address feeRecipient
) public
```

Unwraps the contract's WETH9 balance and sends it to recipient as ETH, with a percentage between 0 (exclusive), and 1 (inclusive) going to `feeRecipient`

The `amountMinimum` parameter prevents malicious contracts from stealing WETH9 from users.

sweepTokenWithFee

```
function sweepTokenWithFee(
    address token,
    uint256 amountMinimum,
    address recipient,
    uint256 feeBips,
    address feeRecipient
) public
```

Transfers the full amount of a token held by this contract to recipient, with a percentage between 0 (exclusive) and 1 (inclusive) going to `feeRecipient`

The `amountMinimum` parameter prevents malicious contracts from stealing the token from users

Functions

createAndInitializePoolIfNecessary

```
function createAndInitializePoolIfNecessary(
    address token0,
    address token1,
    uint24 fee,
```

```

    uint160 sqrtPriceX96
) external returns (address pool)

```

Creates a new pool if it does not exist, then initializes if not initialized

This method can be bundled with others via IMulticall for the first action (e.g. mint) performed against a pool

Parameters:

| Name | Type | Description |
|--------------|---------|---|
| token0 | address | The contract address of token0 of the pool |
| token1 | address | The contract address of token1 of the pool |
| fee | uint24 | The fee amount of the v3 pool for the specified token pair |
| sqrtPriceX96 | uint160 | The initial square root price of the pool as a Q64.96 value |

Return Values:

| Name | Type | Description |
|---|---------|---|
| pool | address | Returns the pool address based on the pair of tokens and fee, will return the newly created pool address if necessary |
| Functionality to call permit on any EIP-2612-compliant token for use in the route | | |

These functions are expected to be embedded in multicalls to allow EOAs to approve a contract and call a function that requires an approval in a single transaction.

Functions

selfPermit

```

function selfPermit(
    address token,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) public

```

Permits this contract to spend a given token from `msg.sender`

The `owner` is always `msg.sender` and the `spender` is always `address(this)`.

Parameters:

| Name | Type | Description |
|----------|---------|---|
| token | address | The address of the token spent |
| value | uint256 | The amount that can be spent of token |
| deadline | uint256 | A timestamp, the current blocktime must be less than or equal to this timestamp |
| v | uint8 | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>s</code> |
| r | bytes32 | Must produce valid secp256k1 signature from the holder along with <code>v</code> and <code>s</code> |
| s | bytes32 | Must produce valid secp256k1 signature from the holder along with <code>v</code> and <code>r</code> |

selfPermitIfNecessary

```

function selfPermitIfNecessary(
    address token,
    uint256 value,
    uint256 deadline,
    uint8 v,

```

```

    bytes32 r,
    bytes32 s
) external

```

Permits this contract to spend a given token from `msg.sender`

The `owner` is always `msg.sender` and the `spender` is always `address(this)`. Can be used instead of `#selfPermit` to prevent calls from failing due to a frontrun of a call to `#selfPermit`

Parameters:

| Name | Type | Description |
|-----------------------|----------------------|---|
| <code>token</code> | <code>address</code> | The address of the token spent |
| <code>value</code> | <code>uint256</code> | The amount that can be spent of token |
| <code>deadline</code> | <code>uint256</code> | A timestamp, the current blocktime must be less than or equal to this timestamp |
| <code>v</code> | <code>uint8</code> | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>s</code> |
| <code>r</code> | <code>bytes32</code> | Must produce valid secp256k1 signature from the holder along with <code>v</code> and <code>s</code> |
| <code>s</code> | <code>bytes32</code> | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>v</code> |

selfPermitAllowed

```

function selfPermitAllowed(
    address token,
    uint256 nonce,
    uint256 expiry,
    uint8 v,
    bytes32 r,
    bytes32 s
) public

```

Permits this contract to spend the sender's tokens for permit signatures that have the `allowed` parameter

The `owner` is always `msg.sender` and the `spender` is always `address(this)`

Parameters:

| Name | Type | Description |
|---------------------|----------------------|---|
| <code>token</code> | <code>address</code> | The address of the token spent |
| <code>nonce</code> | <code>uint256</code> | The current nonce of the owner |
| <code>expiry</code> | <code>uint256</code> | The timestamp at which the permit is no longer valid |
| <code>v</code> | <code>uint8</code> | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>s</code> |
| <code>r</code> | <code>bytes32</code> | Must produce valid secp256k1 signature from the holder along with <code>v</code> and <code>s</code> |
| <code>s</code> | <code>bytes32</code> | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>v</code> |

selfPermitAllowedIfNecessary

```

function selfPermitAllowedIfNecessary(
    address token,
    uint256 nonce,
    uint256 expiry,
    uint8 v,
    bytes32 r,
    bytes32 s
) external

```

Permits this contract to spend the sender's tokens for permit signatures that have the `allowed` parameter

The `owner` is always `msg.sender` and the `spender` is always `address(this)` Can be used instead of `#selfPermitAllowed` to prevent calls from failing due to a frontrun of a call to `#selfPermitAllowed`.

Parameters:

| Name | Type | Description |
|--|---------|---|
| token | address | The address of the token spent |
| nonce | uint256 | The current nonce of the owner |
| expiry | uint256 | The timestamp at which the permit is no longer valid |
| v | uint8 | Must produce valid secp256k1 signature from the holder along with r and s |
| r | bytes32 | Must produce valid secp256k1 signature from the holder along with v and s |
| s | bytes32 | Must produce valid secp256k1 signature from the holder along with r and v |
| Extension to IERC20 that includes token metadata | | |

Functions**name**

```
function name(
) external returns (string)
```

Return Values:

| Type | Description |
|--------|-------------------|
| string | name of the token |

symbol

```
function symbol(
) external returns (string)
```

Return Values:

| Type | Description |
|--------|---------------------|
| string | symbol of the token |

decimals

```
function decimals(
) external returns (uint8)
```

Return Values:

| Type | Description |
|---|--|
| uint8 | number of decimal places the token has |
| Extension to ERC721 that includes a permit function for signature based approvals | |

Functions**PERMIT_TYPEHASH**

```
function PERMIT_TYPEHASH(
) external returns (bytes32)
```

The permit typehash used in the permit signature

Return Values:

| Type | Description |
|------|-------------|
| | |

| | |
|---------|-------------------------|
| bytes32 | typehash for the permit |
|---------|-------------------------|

DOMAIN_SEPARATOR

```
function DOMAIN_SEPARATOR(
) external returns (bytes32)
```

The domain separator used in the permit signature

Return Values:

| Name | Type | Description |
|------|------|---|
| The | | domain seperator used in encoding of permit signature |

permit

```
function permit(
    address spender,
    uint256 tokenId,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external
```

Approve of a specific token ID for spending by spender via signature

Parameters:

| Name | Type | Description |
|---|---------|--|
| spender | address | The account that is being approved |
| tokenId | uint256 | The ID of the token that is being approved for spending |
| deadline | uint256 | The deadline timestamp by which the call must be mined for the approve to work |
| v | uint8 | Must produce valid secp256k1 signature from the holder along with r and s |
| r | bytes32 | Must produce valid secp256k1 signature from the holder along with v and s |
| s | bytes32 | Must produce valid secp256k1 signature from the holder along with r and v |
| Enables calling multiple methods in a single call to the contract | | |

Functions

multicall

```
function multicall(
    bytes[] data
) external returns (bytes[] results)
```

Call multiple functions in the current contract and return the data from all of them if they all succeed

The `msg.value` should not be trusted for any method callable from multicall.

Parameters:

| Name | Type | Description |
|------|---------|--|
| data | bytes[] | The encoded function data for each of the calls to make to this contract |

Return Values:

| Name | Type | Description |
|--|---------|---|
| results | bytes[] | The results from each of the calls passed in via data |
| Wraps Uniswap V3 positions in a non-fungible token interface which allows for them to be transferred and authorized. | | |
| | | |

Parameter Structs**MintParams**

```
struct MintParams {
    address token0;
    address token1;
    uint24 fee;
    int24 tickLower;
    int24 tickUpper;
    uint256 amount0Desired;
    uint256 amount1Desired;
    uint256 amount0Min;
    uint256 amount1Min;
    address recipient;
    uint256 deadline;
}
```

IncreaseLiquidityParams

```
struct IncreaseLiquidityParams {
    uint256 tokenId;
    uint256 amount0Desired;
    uint256 amount1Desired;
    uint256 amount0Min;
    uint256 amount1Min;
    uint256 deadline;
}
```

DecreaseLiquidityParams

```
struct DecreaseLiquidityParams {
    uint256 tokenId;
    uint128 liquidity;
    uint256 amount0Min;
    uint256 amount1Min;
    uint256 deadline;
}
```

CollectParams

```
struct CollectParams {
    uint256 tokenId;
    address recipient;
    uint128 amount0Max;
    uint128 amount1Max;
}
```

Functions**positions**

```

function positions(
    uint256 tokenId
) external view returns (uint96 nonce, address operator, address token0, address token1, uint24 fee, int24 tickLower, int24 tickUpper, uint128 liquidity, uint256 feeGrowthInside0LastX128, uint256 feeGrowthInside1LastX128, uint128 tokensOwed0, uint128 tokensOwed1)

```

Returns the position information associated with a given token ID.

Throws if the token ID is not valid.

Parameters:

| Name | Type | Description |
|---------|---------|--|
| tokenId | uint256 | The ID of the token that represents the position |

Return Values:

| Name | Type | Description |
|--------------------------|---------|--|
| nonce | uint96 | The nonce for permits |
| operator | address | The address that is approved for spending |
| token0 | address | The address of the token0 for a specific pool |
| token1 | address | The address of the token1 for a specific pool |
| fee | uint24 | The fee associated with the pool |
| tickLower | int24 | The lower end of the tick range for the position |
| tickUpper | int24 | The higher end of the tick range for the position |
| liquidity | uint128 | The liquidity of the position |
| feeGrowthInside0LastX128 | uint256 | The fee growth of token0 as of the last action on the individual position |
| feeGrowthInside1LastX128 | uint256 | The fee growth of token1 as of the last action on the individual position |
| tokensOwed0 | uint128 | The uncollected amount of token0 owed to the position as of the last computation |
| tokensOwed1 | uint128 | The uncollected amount of token1 owed to the position as of the last computation |

createAndInitializePoolIfNecessary

```

function createAndInitializePoolIfNecessary(
    address tokenA,
    address tokenB,
    uint24 fee,
    uint160 sqrtPriceX96
) external returns (address pool)

```

Creates a new pool if it does not exist, then initializes if not initialized

This method can be bundled with mint for the first mint of a pool to create, initialize a pool and mint at the same time

Parameters:

| Name | Type | Description |
|--------------|---------|---|
| tokenA | address | The contract address of either token0 or token1 |
| tokenB | address | The contract address of the other token |
| fee | uint24 | The fee amount of the v3 pool for the specified token pair |
| sqrtPriceX96 | uint160 | The initial square root price of the pool as a Q64.96 value |

We use tokenA and tokenB when we are referring to unsorted, or unordered tokens

Return Values:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|------|---------|---|
| pool | address | Returns the pool address based on the pair of tokens and fee, will return the newly created pool address if necessary |
|------|---------|---|

mint

```
function mint(
    struct INonfungiblePositionManager.MintParams params
) external returns (uint256 tokenId, uint128 liquidity, uint256 amount0, uint256 amount1)
```

Creates a new position wrapped in a NFT

Call this when the pool does exist and is initialized. Note that if the pool is created but not initialized a method does not exist, i.e. the pool is assumed to be initialized.

Parameters:

| Name | Type | Description |
|--------|---|---|
| params | struct INonfungiblePositionManager.MintParams | The params necessary to mint a position, encoded as <code>MintParams</code> in calldata |

Return Values:

| Name | Type | Description |
|-----------|---------|---|
| tokenId | uint256 | The ID of the token that represents the minted position |
| liquidity | uint128 | The amount of liquidity for this position |
| amount0 | uint256 | The amount of token0 |
| amount1 | uint256 | The amount of token1 |

increaseLiquidity

```
function increaseLiquidity(
    struct INonfungiblePositionManager.IncreaseLiquidityParams params
) external returns (uint128 liquidity, uint256 amount0, uint256 amount1)
```

Increases the amount of liquidity in a position, with tokens paid by the `msg.sender`

Parameters:

| Name | Type | Description |
|--------|--|---|
| params | struct INonfungiblePositionManager.IncreaseLiquidityParams | tokenId The ID of the token for which liquidity is being increased, |

Return Values:

| Name | Type | Description |
|-----------|---------|--|
| liquidity | uint128 | The new liquidity amount as a result of the increase |
| amount0 | uint256 | The amount of token0 to achieve resulting liquidity |
| amount1 | uint256 | The amount of token1 to achieve resulting liquidity |

decreaseLiquidity

```
function decreaseLiquidity(
    struct INonfungiblePositionManager.DecreaseLiquidityParams params
) external returns (uint256 amount0, uint256 amount1)
```

Decreases the amount of liquidity in a position and accounts it to the position

Parameters:

| Name | Type | Description |
|--------|--|--|
| params | struct INonfungiblePositionManager.DecreaseLiquidityParams | tokenId The ID of the token for which liquidity is being decreased |

Return Values:

| Name | Type | Description |
|---------|---------|--|
| amount0 | uint256 | The amount of token0 sent to recipient |
| amount1 | uint256 | The amount of token1 sent to recipient |

collect

```
function collect(
    struct INonfungiblePositionManager.CollectParams params
) external returns (uint256 amount0, uint256 amount1)
```

Collects up to a maximum amount of fees owed to a specific position to the recipient

Parameters:

| Name | Type | Description |
|--------|--|---|
| params | struct INonfungiblePositionManager.CollectParams | tokenId The ID of the NFT for which tokens are being collected, |

Return Values:

| Name | Type | Description |
|---------|---------|--|
| amount0 | uint256 | The amount of fees collected in token0 |
| amount1 | uint256 | The amount of fees collected in token1 |

burn

```
function burn(
    uint256 tokenId
) external
```

Burns a token ID, which deletes it from the NFT contract. The token must have 0 liquidity and all tokens must be collected first.

Parameters:

| Name | Type | Description |
|---------|---------|--|
| tokenId | uint256 | The ID of the token that is being burned |

Events

IncreaseLiquidity

```
event IncreaseLiquidity(
    uint256 tokenId,
    uint128 liquidity,
    uint256 amount0,
    uint256 amount1
)
```

Emitted when liquidity is increased for a position NFT

Also emitted when a token is minted

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| tokenId | uint256 | The ID of the token for which liquidity was increased |
| liquidity | uint128 | The amount by which liquidity for the NFT position was increased |
| amount0 | uint256 | The amount of token0 that was paid for the increase in liquidity |
| amount1 | uint256 | The amount of token1 that was paid for the increase in liquidity |

DecreaseLiquidity

```
event DecreaseLiquidity(
    uint256 tokenId,
    uint128 liquidity,
    uint256 amount0,
    uint256 amount1
)
```

Emitted when liquidity is decreased for a position NFT

Parameters:

| Name | Type | Description |
|-----------|---------|---|
| tokenId | uint256 | The ID of the token for which liquidity was decreased |
| liquidity | uint128 | The amount by which liquidity for the NFT position was decreased |
| amount0 | uint256 | The amount of token0 that was accounted for the decrease in liquidity |
| amount1 | uint256 | The amount of token1 that was accounted for the decrease in liquidity |

Collect

```
event Collect(
    uint256 tokenId,
    address recipient,
    uint256 amount0,
    uint256 amount1
)
```

Emitted when tokens are collected for a position NFT

The amounts reported may not be exactly equivalent to the amounts transferred, due to rounding behavior

Parameters:

| Name | Type | Description |
|-----------|---------|--|
| tokenId | uint256 | The ID of the token for which underlying tokens were collected |
| recipient | address | The address of the account that received the collected tokens |
| amount0 | uint256 | The amount of token0 owed to the position that was collected |
| amount1 | uint256 | The amount of token1 owed to the position that was collected |

Functions

tokenURI

```
function tokenURI(
    contract INonfungiblePositionManager positionManager,
    uint256 tokenId
) external returns (string)
```

Produces the URI describing a particular token ID for a position manager

Note this URI may be a data: URI with the JSON contents directly inlined

Parameters:

| Name | Type | Description |
|-----------------|--------------------------------------|--|
| positionManager | contract INonfungiblePositionManager | The position manager for which to describe the token |
| tokenId | uint256 | The ID of the token for which to produce a description, which may not be valid |

Return Values:

| Type | Description |
|--------|--------------------------------------|
| string | URI of the ERC721-compliant metadata |

Events

UpdateTokenRatioPriority

```
event UpdateTokenRatioPriority(
    address token,
    int256 priority
)
```

Emitted when a token is given a new priority order in the displayed price ratio

Parameters:

| Name | Type | Description |
|---|---------|---|
| token | address | The token being given priority order |
| priority | int256 | Represents priority in ratio - higher integers get numerator priority |
| Functions that return immutable state of the router | | |

Functions

factory

```
function factory(
) external returns (address)
```

Return Values:

| Type | Description |
|---------|---------------------------------------|
| address | the address of the Uniswap V3 factory |

WETH9

```
function WETH9(
) external returns (address)
```

Return Values:

| Type | Description |
|---|----------------------|
| address | the address of WETH9 |
| Functions to ease deposits and withdrawals of ETH | |

Functions

unwrapWETH9

```
function unwrapWETH9(
    uint256 amountMinimum,
    address recipient
) external
```

Unwraps the contract's WETH9 balance and sends it to recipient as ETH.

The amountMinimum parameter prevents malicious contracts from stealing WETH9 from users.

Parameters:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|---------------|---------|---------------------------------------|
| amountMinimum | uint256 | The minimum amount of WETH9 to unwrap |
| recipient | address | The address receiving ETH |

refundETH

```
function refundETH(
) external
```

Refunds any ETH balance held by this contract to the `msg.sender`

Useful for bundling with mint or increase liquidity that uses ether, or exact output swaps that use ether for the input amount

sweepToken

```
function sweepToken(
address token,
uint256 amountMinimum,
address recipient
) external
```

Transfers the full amount of a token held by this contract to recipient

The `amountMinimum` parameter prevents malicious contracts from stealing the token from users

Parameters:

| Name | Type | Description |
|---|---------|---|
| token | address | The contract address of the token which will be transferred to <code>recipient</code> |
| amountMinimum | uint256 | The minimum amount of token required for a transfer |
| recipient | address | The destination address of the token |
| Functions to ease deposits and withdrawals of ETH | | |

Functions

unwrapWETH9WithFee

```
function unwrapWETH9WithFee(
) external
```

Unwraps the contract's WETH9 balance and sends it to recipient as ETH, with a percentage between 0 (exclusive), and 1 (inclusive) going to `feeRecipient`

The `amountMinimum` parameter prevents malicious contracts from stealing WETH9 from users.

sweepTokenWithFee

```
function sweepTokenWithFee(
) external
```

Transfers the full amount of a token held by this contract to recipient, with a percentage between 0 (exclusive) and 1 (inclusive) going to `feeRecipient`

The `amountMinimum` parameter prevents malicious contracts from stealing the token from users Provides a method for creating and initializing a pool, if necessary, for bundling with other methods that require the pool to exist.

Functions

createAndInitializePoolIfNecessary

```
function createAndInitializePoolIfNecessary(
address token0,
address token1,
```

```

        uint24 fee,
        uint160 sqrtPriceX96
    ) external returns (address pool)

```

Creates a new pool if it does not exist, then initializes if not initialized

This method can be bundled with others via IMulticall for the first action (e.g. mint) performed against a pool

Parameters:

| Name | Type | Description |
|--------------|---------|---|
| token0 | address | The contract address of token0 of the pool |
| token1 | address | The contract address of token1 of the pool |
| fee | uint24 | The fee amount of the v3 pool for the specified token pair |
| sqrtPriceX96 | uint160 | The initial square root price of the pool as a Q64.96 value |

Return Values:

| Name | Type | Description |
|--|---------|---|
| pool | address | Returns the pool address based on the pair of tokens and fee, will return the newly created pool address if necessary |
| Supports quoting the calculated amounts from exact input or exact output swaps | | |

These functions are not marked view because they rely on calling non-view functions and reverting to compute the result. They are also not gas efficient and should not be called on-chain.

Functions

quoteExactInput

```

function quoteExactInput(
    bytes path,
    uint256 amountIn
) external returns (uint256 amountOut)

```

Returns the amount out received for a given exact input swap without executing the swap

Parameters:

| Name | Type | Description |
|----------|---------|---|
| path | bytes | The path of the swap, i.e. each token pair and the pool fee |
| amountIn | uint256 | The amount of the first token to swap |

Return Values:

| Name | Type | Description |
|-----------|---------|---|
| amountOut | uint256 | The amount of the last token that would be received |

quoteExactInputSingle

```

function quoteExactInputSingle(
    address tokenIn,
    address tokenOut,
    uint24 fee,
    uint256 amountIn,
    uint160 sqrtPriceLimitX96
) external returns (uint256 amountOut)

```

Returns the amount out received for a given exact input but for a swap of a single pool

Parameters:

| Name | Type | Description |
|-------------------|---------|---|
| tokenIn | address | The token being swapped in |
| tokenOut | address | The token being swapped out |
| fee | uint24 | The fee of the token pool to consider for the pair |
| amountIn | uint256 | The desired input amount |
| sqrtPriceLimitX96 | uint160 | The price limit of the pool that cannot be exceeded by the swap |

Return Values:

| Name | Type | Description |
|-----------|---------|--|
| amountOut | uint256 | The amount of <code>tokenOut</code> that would be received |

quoteExactOutput

```
function quoteExactOutput(
    bytes path,
    uint256 amountOut
) external returns (uint256 amountIn)
```

Returns the amount in required for a given exact output swap without executing the swap

Parameters:

| Name | Type | Description |
|-----------|---------|---|
| path | bytes | The path of the swap, i.e. each token pair and the pool fee |
| amountOut | uint256 | The amount of the last token to receive |

Return Values:

| Name | Type | Description |
|----------|---------|---|
| amountIn | uint256 | The amount of first token required to be paid |

quoteExactOutputSingle

```
function quoteExactOutputSingle(
    address tokenIn,
    address tokenOut,
    uint24 fee,
    uint256 amountOut,
    uint160 sqrtPriceLimitX96
) external returns (uint256 amountIn)
```

Returns the amount in required to receive the given exact output amount but for a swap of a single pool

Parameters:

| Name | Type | Description |
|-------------------|---------|---|
| tokenIn | address | The token being swapped in |
| tokenOut | address | The token being swapped out |
| fee | uint24 | The fee of the token pool to consider for the pair |
| amountOut | uint256 | The desired output amount |
| sqrtPriceLimitX96 | uint160 | The price limit of the pool that cannot be exceeded by the swap |

Return Values:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|---|---------|--|
| amountIn | uint256 | The amount required as the input for the swap in order to receive <code>amountOut</code> |
| Supports quoting the calculated amounts from exact input or exact output swaps. | | |
| For each pool also tells you the number of initialized ticks crossed and the sqrt price of the pool after the swap. | | |

These functions are not marked view because they rely on calling non-view functions and reverting to compute the result. They are also not gas efficient and should not be called on-chain.

Functions

quoteExactInput

```
function quoteExactInput(
    bytes path,
    uint256 amountIn
) external returns (uint256 amountOut, uint160[] sqrtPriceX96AfterList, uint32[] initializedTicksCrossedList,
    uint256 gasEstimate)
```

Returns the amount out received for a given exact input swap without executing the swap

Parameters:

| Name | Type | Description |
|----------|---------|---|
| path | bytes | The path of the swap, i.e. each token pair and the pool fee |
| amountIn | uint256 | The amount of the first token to swap |

Return Values:

| Name | Type | Description |
|-----------------------------|---------|---|
| amountOut | bytes | The amount of the last token that would be received |
| sqrtPriceX96AfterList | uint256 | List of the sqrt price after the swap for each pool in the path |
| initializedTicksCrossedList | | List of the initialized ticks that the swap crossed for each pool in the path |
| gasEstimate | | The estimate of the gas that the swap consumes |

quoteExactInputSingle

```
function quoteExactInputSingle(
    struct IQuoterV2.QuoteExactInputSingleParams params
) external returns (uint256 amountOut, uint160 sqrtPriceX96After, uint32 initializedTicksCrossed, uint256
gasEstimate)
```

Returns the amount out received for a given exact input but for a swap of a single pool

Parameters:

| Name | Type | Description |
|--------|--|---|
| params | struct IQuoterV2.QuoteExactInputSingleParams | The params for the quote, encoded as <code>QuoteExactInputSingleParams</code> |

`tokenIn` The token being swapped in
`tokenOut` The token being swapped out
`fee` The fee of the token pool to consider for the pair
`amountIn` The desired input amount
`sqrtPriceLimitX96` The price limit of the pool that cannot be exceeded by the swap

Return Values:

| Name | Type | Description |
|-------------------------|--|--|
| amountOut | struct IQuoterV2.QuoteExactInputSingleParams | The amount of <code>tokenOut</code> that would be received |
| sqrtPriceX96After | | The sqrt price of the pool after the swap |
| initializedTicksCrossed | | The number of initialized ticks that the swap crossed |

| | | |
|-------------|--|--|
| gasEstimate | | The estimate of the gas that the swap consumes |
|-------------|--|--|

quoteExactOutput

```
function quoteExactOutput(
    bytes path,
    uint256 amountOut
) external returns (uint256 amountIn, uint160[] sqrtPriceX96AfterList, uint32[] initializedTicksCrossedList,
uint256 gasEstimate)
```

Returns the amount in required for a given exact output swap without executing the swap

Parameters:

| Name | Type | Description |
|-----------|---------|---|
| path | bytes | The path of the swap, i.e. each token pair and the pool fee. Path must be provided in reverse order |
| amountOut | uint256 | The amount of the last token to receive |

Return Values:

| Name | Type | Description |
|-----------------------------|---------|---|
| amountIn | bytes | The amount of first token required to be paid |
| sqrtPriceX96AfterList | uint256 | List of the sqrt price after the swap for each pool in the path |
| initializedTicksCrossedList | | List of the initialized ticks that the swap crossed for each pool in the path |
| gasEstimate | | The estimate of the gas that the swap consumes |

quoteExactOutputSingle

```
function quoteExactOutputSingle(
    struct IQuoterV2.QuoteExactOutputSingleParams params
) external returns (uint256 amountIn, uint160 sqrtPriceX96After, uint32 initializedTicksCrossed, uint256
gasEstimate)
```

Returns the amount in required to receive the given exact output amount but for a swap of a single pool

Parameters:

| Name | Type | Description |
|--------|---|--|
| params | struct IQuoterV2.QuoteExactOutputSingleParams | The params for the quote, encoded as <code>QuoteExactOutputSingleParams</code> |

tokenIn The token being swapped in
tokenOut The token being swapped out
fee The fee of the token pool to consider for the pair
amountOut The desired output amount
sqrtPriceLimitX96 The price limit of the pool that cannot be exceeded by the swap

Return Values:

| Name | Type | Description |
|---|---|--|
| amountIn | struct IQuoterV2.QuoteExactOutputSingleParams | The amount required as the input for the swap in order to receive <code>amountOut</code> |
| sqrtPriceX96After | | The sqrt price of the pool after the swap |
| initializedTicksCrossed | | The number of initialized ticks that the swap crossed |
| gasEstimate | | The estimate of the gas that the swap consumes |
| Functionality to call permit on any EIP-2612-compliant token for use in the route | | |

Functions

selfPermit

```
function selfPermit(
    address token,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external
```

Permits this contract to spend a given token from `msg.sender`

The `owner` is always `msg.sender` and the `spender` is always `address(this)`.

Parameters:

| Name | Type | Description |
|-----------------------|---------|---|
| <code>token</code> | address | The address of the token spent |
| <code>value</code> | uint256 | The amount that can be spent of token |
| <code>deadline</code> | uint256 | A timestamp, the current blocktime must be less than or equal to this timestamp |
| <code>v</code> | uint8 | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>s</code> |
| <code>r</code> | bytes32 | Must produce valid secp256k1 signature from the holder along with <code>v</code> and <code>s</code> |
| <code>s</code> | bytes32 | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>v</code> |

selfPermitIfNecessary

```
function selfPermitIfNecessary(
    address token,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external
```

Permits this contract to spend a given token from `msg.sender`

The `owner` is always `msg.sender` and the `spender` is always `address(this)`. Can be used instead of `#selfPermit` to prevent calls from failing due to a frontrun of a call to `#selfPermit`

Parameters:

| Name | Type | Description |
|-----------------------|---------|---|
| <code>token</code> | address | The address of the token spent |
| <code>value</code> | uint256 | The amount that can be spent of token |
| <code>deadline</code> | uint256 | A timestamp, the current blocktime must be less than or equal to this timestamp |
| <code>v</code> | uint8 | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>s</code> |
| <code>r</code> | bytes32 | Must produce valid secp256k1 signature from the holder along with <code>v</code> and <code>s</code> |
| <code>s</code> | bytes32 | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>v</code> |

selfPermitAllowed

```
function selfPermitAllowed(
    address token,
    uint256 nonce,
    uint256 expiry,
    uint8 v,
    bytes32 r,
```

```
    bytes32 s
) external
```

Permits this contract to spend the sender's tokens for permit signatures that have the `allowed` parameter

The `owner` is always `msg.sender` and the `spender` is always `address(this)`

Parameters:

| Name | Type | Description |
|---------------------|----------------------|---|
| <code>token</code> | <code>address</code> | The address of the token spent |
| <code>nonce</code> | <code>uint256</code> | The current nonce of the owner |
| <code>expiry</code> | <code>uint256</code> | The timestamp at which the permit is no longer valid |
| <code>v</code> | <code>uint8</code> | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>s</code> |
| <code>r</code> | <code>bytes32</code> | Must produce valid secp256k1 signature from the holder along with <code>v</code> and <code>s</code> |
| <code>s</code> | <code>bytes32</code> | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>v</code> |

`selfPermitAllowedIfNecessary`

```
function selfPermitAllowedIfNecessary(
    address token,
    uint256 nonce,
    uint256 expiry,
    uint8 v,
    bytes32 r,
    bytes32 s
) external
```

Permits this contract to spend the sender's tokens for permit signatures that have the `allowed` parameter

The `owner` is always `msg.sender` and the `spender` is always `address(this)` Can be used instead of `#selfPermitAllowed` to prevent calls from failing due to a frontrun of a call to `#selfPermitAllowed`.

Parameters:

| Name | Type | Description |
|--|----------------------|---|
| <code>token</code> | <code>address</code> | The address of the token spent |
| <code>nonce</code> | <code>uint256</code> | The current nonce of the owner |
| <code>expiry</code> | <code>uint256</code> | The timestamp at which the permit is no longer valid |
| <code>v</code> | <code>uint8</code> | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>s</code> |
| <code>r</code> | <code>bytes32</code> | Must produce valid secp256k1 signature from the holder along with <code>v</code> and <code>s</code> |
| <code>s</code> | <code>bytes32</code> | Must produce valid secp256k1 signature from the holder along with <code>r</code> and <code>v</code> |
| Functions for swapping tokens via Uniswap V3 | | |

Parameter Structs

Note that `fee` is in hundredths of basis points (e.g. the `fee` for a pool at the 0.3% tier is 3000; the `fee` for a pool at the 0.01% tier is 100).

`ExactInputSingleParams`

```
struct ExactInputSingleParams {
    address tokenIn;
    address tokenOut;
    uint24 fee;
    address recipient;
    uint256 deadline;
    uint256 amountIn;
    uint256 amountOutMinimum;
```

```
        uint160 sqrtPriceLimitX96;
    }
```

ExactInputParams

```
struct ExactInputParams {
    bytes path;
    address recipient;
    uint256 deadline;
    uint256 amountIn;
    uint256 amountOutMinimum;
}
```

ExactOutputSingleParams

```
struct ExactOutputSingleParams {
    address tokenIn;
    address tokenOut;
    uint24 fee;
    address recipient;
    uint256 deadline;
    uint256 amountOut;
    uint256 amountInMaximum;
    uint160 sqrtPriceLimitX96;
}
```

ExactOutputParams

```
struct ExactOutputParams {
    bytes path;
    address recipient;
    uint256 deadline;
    uint256 amountOut;
    uint256 amountInMaximum;
}
```

Functions

exactInputSingle

```
function exactInputSingle(
    struct ISwapRouter.ExactInputSingleParams params
) external returns (uint256 amountOut)
```

Swaps `amountIn` of one token for as much as possible of another token

Parameters:

| Name | Type | Description |
|--------|---|---|
| params | struct ISwapRouter.ExactInputSingleParams | The parameters necessary for the swap, encoded as <code>ExactInputSingleParams</code> in calldata |

Return Values:

| Name | Type | Description |
|-----------|---|----------------------------------|
| amountOut | struct ISwapRouter.ExactInputSingleParams | The amount of the received token |

exactInput

```
function exactInput(
    struct ISwapRouter.ExactInputParams params
) external returns (uint256 amountOut)
```

Swaps `amountIn` of one token for as much as possible of another along the specified path

Parameters:

| Name | Type | Description |
|---------------------|---|---|
| <code>params</code> | struct
<code>ISwapRouter.ExactInputParams</code> | The parameters necessary for the multi-hop swap, encoded as <code>ExactInputParams</code> in calldata |

Return Values:

| Name | Type | Description |
|------------------------|--|----------------------------------|
| <code>amountOut</code> | struct <code>ISwapRouter.ExactInputParams</code> | The amount of the received token |

exactOutputSingle

```
function exactOutputSingle(
    struct ISwapRouter.ExactOutputSingleParams params
) external returns (uint256 amountIn)
```

Swaps as little as possible of one token for `amountOut` of another token

Parameters:

| Name | Type | Description |
|---------------------|--|--|
| <code>params</code> | struct
<code>ISwapRouter.ExactOutputSingleParams</code> | The parameters necessary for the swap, encoded as <code>ExactOutputSingleParams</code> in calldata |

Return Values:

| Name | Type | Description |
|-----------------------|---|-------------------------------|
| <code>amountIn</code> | struct <code>ISwapRouter.ExactOutputSingleParams</code> | The amount of the input token |

exactOutput

```
function exactOutput(
    struct ISwapRouter.ExactOutputParams params
) external returns (uint256 amountIn)
```

Swaps as little as possible of one token for `amountOut` of another along the specified path (reversed)

Parameters:

| Name | Type | Description |
|---------------------|--|--|
| <code>params</code> | struct
<code>ISwapRouter.ExactOutputParams</code> | The parameters necessary for the multi-hop swap, encoded as <code>ExactOutputParams</code> in calldata |

Return Values:

| Name | Type | Description |
|--|---|-------------------------------|
| <code>amountIn</code> | struct <code>ISwapRouter.ExactOutputParams</code> | The amount of the input token |
| Provides functions for fetching chunks of tick data for a pool | | |

This avoids the waterfall of fetching the tick bitmap, parsing the bitmap to know which ticks to fetch, and then sending additional multicalls to fetch the tick data

Functions

getPopulatedTicksInWord

```
function getPopulatedTicksInWord(
    address pool,
```

```

int16 tickBitmapIndex
) external returns (struct ITickLens.PopulatedTick[] populatedTicks)

```

Get all the tick data for the populated ticks from a word of the tick bitmap of a pool

Parameters:

| Name | Type | Description |
|-----------------|---------|--|
| pool | address | The address of the pool for which to fetch populated tick data |
| tickBitmapIndex | int16 | The index of the word in the tick bitmap for which to parse the bitmap and fetch all the populated ticks |

Return Values:

| Name | Type | Description |
|---|---------------------------|---|
| populatedTicks | ITickLens.PopulatedTick[] | An array of tick data for the given word in the tick bitmap |
| Enables migration of liquidity from Uniswap v2-compatible pairs into Uniswap v3 pools | | |

Functions

migrate

```

function migrate(
    struct IV3Migrator.MigrateParams params
) external

```

Migrates liquidity to v3 by burning v2 liquidity and minting a new position for v3

Slippage protection is enforced via `amount{0,1}Min`, which should be a discount of the expected values of the maximum amount of v3 liquidity that the v2 liquidity can get. For the special case of migrating to an out-of-range position, `amount{0,1}Min` may be set to 0, enforcing that the position remains out of range

Parameters:

| Name | Type | Description |
|---|----------------------------------|---|
| params | struct IV3Migrator.MigrateParams | The params necessary to migrate v2 liquidity, encoded as <code>MigrateParams</code> in calldata |
| Interface that verifies provided signature for the data | | |

Interface defined by EIP-1271

Functions

isValidSignature

```

function isValidSignature(
    bytes32 hash,
    bytes signature
) external returns (bytes4 magicValue)

```

Returns whether the provided signature is valid for the provided data

MUST return the bytes4 magic value `0x1626ba7e` when function passes. MUST NOT modify state (using `STATICCALL` for `solc < 0.5`, `view` modifier for `solc > 0.5`). MUST allow external calls.

Parameters:

| Name | Type | Description |
|-----------|---------|---|
| hash | bytes32 | Hash of the data to be signed |
| signature | bytes | Signature byte array associated with <code>_data</code> |

Return Values:

| Name | Type | Description |
|---------------------------------------|---------|-----------------------------------|
| magicValue | bytes32 | The bytes4 magic value 0x1626ba7e |
| Interface used by DAI/CHAI for permit | | |

Functions**permit**

```
function permit(
    address holder,
    address spender,
    uint256 nonce,
    uint256 expiry,
    bool allowed,
    uint8 v,
    bytes32 r,
    bytes32 s
) external
```

Approve the spender to spend some tokens via the holder signature

This is the permit interface used by DAI and CHAI

Parameters:

| Name | Type | Description |
|---------|---------|---|
| holder | address | The address of the token holder, the token owner |
| spender | address | The address of the token spender |
| nonce | uint256 | The holder's nonce, increases at each call to permit |
| expiry | uint256 | The timestamp at which the permit is no longer valid |
| allowed | bool | Boolean that sets approval amount, true for type(uint256).max and false for 0 |
| v | uint8 | Must produce valid secp256k1 signature from the holder along with r and s |
| r | bytes32 | Must produce valid secp256k1 signature from the holder along with v and s |
| s | bytes32 | Must produce valid secp256k1 signature from the holder along with r and v |

Functions**deposit**

```
function deposit(
) external
```

Deposit ether to get wrapped ether

withdraw

```
function withdraw(
) external
```

Withdraw wrapped ether to get ether Allows getting the expected amount out or amount in for a given swap without executing the swap

These functions are not gas efficient and should *not* be called on chain. Instead, optimistically execute the swap and check the amounts in the callback.

Functions**constructor**

```
function constructor()
) public
```

uniswapV3SwapCallback

```
function uniswapV3SwapCallback(
int256 amount0Delta,
int256 amount1Delta,
bytes data
) external
```

Called to `msg.sender` after executing a swap via `IUniswapV3Pool#swap`.

In the implementation you must pay the pool tokens owed for the swap. The caller of this method must be checked to be a `UniswapV3Pool` deployed by the canonical `UniswapV3Factory`. `amount0Delta` and `amount1Delta` can both be 0 if no tokens were swapped.

Parameters:

| Name | Type | Description |
|---------------------------|---------------------|---|
| <code>amount0Delta</code> | <code>int256</code> | The amount of token0 that was sent (negative) or must be received (positive) by the pool by the end of the swap. If positive, the callback must send that amount of token0 to the pool. <code>amount1Delta</code> <code>int256</code> The amount of token1 that was sent (negative) or must be received (positive) by the pool by the end of the swap. If positive, the callback must send that amount of token1 to the pool. <code>data</code> <code>bytes</code> Any data passed through by the caller via the <code>IUniswapV3PoolActions#swap</code> call |

quoteExactInputSingle

```
function quoteExactInputSingle(
address tokenIn,
address tokenOut,
uint24 fee,
uint256 amountIn,
uint160 sqrtPriceLimitX96
) public returns (uint256 amountOut)
```

Returns the amount out received for a given exact input but for a swap of a single pool

Parameters:

| Name | Type | Description |
|--------------------------------|----------------------|---|
| <code>tokenIn</code> | <code>address</code> | The token being swapped in |
| <code>tokenOut</code> | <code>address</code> | The token being swapped out |
| <code>fee</code> | <code>uint24</code> | The fee of the token pool to consider for the pair |
| <code>amountIn</code> | <code>uint256</code> | The desired input amount |
| <code>sqrtPriceLimitX96</code> | <code>uint160</code> | The price limit of the pool that cannot be exceeded by the swap |

Return Values:

| Name | Type | Description |
|------------------------|----------------------|--|
| <code>amountOut</code> | <code>uint256</code> | The amount of <code>tokenOut</code> that would be received |

quoteExactInput

```
function quoteExactInput(
bytes path,
uint256 amountIn
) external returns (uint256 amountOut)
```

Returns the amount out received for a given exact input swap without executing the swap

Parameters:

| Name | Type | Description |
|----------|---------|---|
| path | bytes | The path of the swap, i.e. each token pair and the pool fee |
| amountIn | uint256 | The amount of the first token to swap |

Return Values:

| Name | Type | Description |
|-----------|---------|---|
| amountOut | uint256 | The amount of the last token that would be received |

quoteExactOutputSingle

```
function quoteExactOutputSingle(
    address tokenIn,
    address tokenOut,
    uint24 fee,
    uint256 amountOut,
    uint160 sqrtPriceLimitX96
) public returns (uint256 amountIn)
```

Returns the amount in required to receive the given exact output amount but for a swap of a single pool

Parameters:

| Name | Type | Description |
|-------------------|---------|---|
| tokenIn | address | The token being swapped in |
| tokenOut | address | The token being swapped out |
| fee | uint24 | The fee of the token pool to consider for the pair |
| amountOut | uint256 | The desired output amount |
| sqrtPriceLimitX96 | uint160 | The price limit of the pool that cannot be exceeded by the swap |

Return Values:

| Name | Type | Description |
|----------|---------|--|
| amountIn | uint256 | The amount required as the input for the swap in order to receive <code>amountOut</code> |

quoteExactOutput

```
function quoteExactOutput(
    bytes path,
    uint256 amountOut
) external returns (uint256 amountIn)
```

Returns the amount in required for a given exact output swap without executing the swap

Parameters:

| Name | Type | Description |
|-----------|---------|---|
| path | bytes | The path of the swap, i.e. each token pair and the pool fee |
| amountOut | uint256 | The amount of the last token to receive |

Return Values:

| Name | Type | Description |
|---|---------|---|
| amountIn | uint256 | The amount of first token required to be paid |
| Allows getting the expected amount out or amount in for a given swap without executing the swap | | |

These functions are not gas efficient and should *not* be called on chain. Instead, optimistically execute the swap and check the amounts in the callback.

Functions

constructor

```
function constructor()
) public
```

uniswapV3SwapCallback

```
function uniswapV3SwapCallback(
int256 amount0Delta,
int256 amount1Delta,
bytes data
) external view override
```

Called to `msg.sender` after executing a swap via `IUniswapV3Pool#swap`.

In the implementation you must pay the pool tokens owed for the swap. The caller of this method must be checked to be a `UniswapV3Pool` deployed by the canonical `UniswapV3Factory`. `amount0Delta` and `amount1Delta` can both be 0 if no tokens were swapped.

Parameters:

| Name | Type | Description |
|---------------------------|---------------------|---|
| <code>amount0Delta</code> | <code>int256</code> | The amount of token0 that was sent (negative) or must be received (positive) by the pool by |

the end of the swap. If positive, the callback must send that amount of token0 to the pool. | `amount1Delta` | `int256` | The amount of token1 that was sent (negative) or must be received (positive) by the pool by the end of the swap. If positive, the callback must send that amount of token1 to the pool. | `data` | `bytes` | Any data passed through by the caller via the `IUniswapV3PoolActions#swap` call

quoteExactInputSingle

```
function quoteExactInputSingle(
) public override returns (uint256 amountOut, uint160 sqrtPriceX96After, uint32 initializedTicksCrossed, uint256
gasEstimate)
```

quoteExactInput

```
function quoteExactInput(
) public override returns (uint256 amountOut, uint160[] sqrtPriceX96AfterList, uint32[]
initializedTicksCrossedList, uint256 gasEstimate)
```

quoteExactOutputSingle

```
function quoteExactOutputSingle(
) public override returns (uint256 amountIn, uint160 sqrtPriceX96After, uint32 initializedTicksCrossed, uint256
gasEstimate)
```

quoteExactOutput

```
function quoteExactOutput(
) public override returns (uint256 amountIn, uint160[] sqrtPriceX96AfterList, uint32[]
initializedTicksCrossedList, uint256 gasEstimate)
```

Functions

getPopulatedTicksInWord

```
function getPopulatedTicksInWord(
address pool,
```

```

int16 tickBitmapIndex
) public returns (struct ITickLens.PopulatedTick[] populatedTicks)

```

Get all the tick data for the populated ticks from a word of the tick bitmap of a pool

Parameters:

| Name | Type | Description |
|-----------------|---------|--|
| pool | address | The address of the pool for which to fetch populated tick data |
| tickBitmapIndex | int16 | The index of the word in the tick bitmap for which to parse the bitmap and fetch all the populated ticks |

Return Values:

| Name | Type | Description |
|---|---------------------------|---|
| populatedTicks | ITickLens.PopulatedTick[] | An array of tick data for the given word in the tick bitmap |
| Provides a function for encoding some bytes in base64 | | |

Functions

encode

```

function encode(
) internal returns (string)

```

Encodes some bytes to the base64 representation

Functions

slice

```

function slice(
) internal returns (bytes)

```

toAddress

```

function toAddress(
) internal returns (address)

```

toUInt24

```

function toUInt24(
) internal returns (uint24)

```

Provides validation for callbacks from Uniswap V3 Pools

Functions

verifyCallback

```

function verifyCallback(
    address factory,
    address tokenA,
    address tokenB,
    uint24 fee
) internal returns (contract IUniswapV3Pool pool)

```

Returns the address of a valid Uniswap V3 Pool

Parameters:

| Name | Type | Description |
|---------|---------|---|
| factory | address | The contract address of the Uniswap V3 factory |
| tokenA | address | The contract address of either token0 or token1 |
| tokenB | address | The contract address of the other token |
| fee | uint24 | The fee collected upon every swap in the pool, denominated in hundredths of a bip |

Return Values:

| Name | Type | Description |
|------|----------------|------------------------------|
| pool | IUniswapV3Pool | The V3 pool contract address |

verifyCallback

```
function verifyCallback(
    address factory,
    struct PoolAddress.PoolKey poolKey
) internal returns (contract IUniswapV3Pool pool)
```

Returns the address of a valid Uniswap V3 Pool

Parameters:

| Name | Type | Description |
|---------|----------------------------|--|
| factory | address | The contract address of the Uniswap V3 factory |
| poolKey | struct PoolAddress.PoolKey | The identifying key of the V3 pool |

Return Values:

| Name | Type | Description |
|------|----------------|------------------------------|
| pool | IUniswapV3Pool | The V3 pool contract address |

Functions

get

```
function get(
) internal returns (uint256 chainId)
```

Gets the current chain ID

Return Values:

| Name | Type | Description |
|---------|---------|----------------------|
| chainId | uint256 | The current chain ID |

Functions

toHexString

```
function toHexString(
) internal returns (string)
```

Converts a `uint256` to its ASCII `string` hexadecimal representation with fixed length.

Credit to Open Zeppelin under MIT license <https://github.com/OpenZeppelin/openzeppelin-contracts/blob/243adff49ce1700e0ecb99fe522fb16cff1d1ddc/contracts/utils/Strings.sol#L55>

toHexStringNoPrefix

```
function toHexStringNoPrefix(
) internal returns (string)
```

Provides functions for computing liquidity amounts from token amounts and prices

Functions

getLiquidityForAmount0

```
function getLiquidityForAmount0(
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    uint256 amount0
) internal returns (uint128 liquidity)
```

Computes the amount of liquidity received for a given amount of token0 and price range

Calculates $\text{amount0} \cdot (\sqrt{\text{upper}} - \sqrt{\text{lower}}) / (\sqrt{\text{upper}} - \sqrt{\text{lower}})$.

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioAX96 | uint160 | A sqrt price representing the first tick boundary |
| sqrtRatioBX96 | uint160 | A sqrt price representing the second tick boundary |
| amount0 | uint256 | The amount0 being sent in |

Return Values:

| Name | Type | Description |
|-----------|---------|----------------------------------|
| liquidity | uint128 | The amount of returned liquidity |

getLiquidityForAmount1

```
function getLiquidityForAmount1(
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    uint256 amount1
) internal returns (uint128 liquidity)
```

Computes the amount of liquidity received for a given amount of token1 and price range

Calculates $\text{amount1} / (\sqrt{\text{upper}} - \sqrt{\text{lower}})$.

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioAX96 | uint160 | A sqrt price representing the first tick boundary |
| sqrtRatioBX96 | uint160 | A sqrt price representing the second tick boundary |
| amount1 | uint256 | The amount1 being sent in |

Return Values:

| Name | Type | Description |
|-----------|---------|----------------------------------|
| liquidity | uint128 | The amount of returned liquidity |

getLiquidityForAmounts

```
function getLiquidityForAmounts(
    uint160 sqrtRatioX96,
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    uint256 amount0,
```

```

    uint256 amount1
) internal returns (uint128 liquidity)

```

Computes the maximum amount of liquidity received for a given amount of token0, token1, the current pool prices and the prices at the tick boundaries

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioX96 | uint160 | A sqrt price representing the current pool prices |
| sqrtRatioAX96 | uint160 | A sqrt price representing the first tick boundary |
| sqrtRatioBX96 | uint160 | A sqrt price representing the second tick boundary |
| amount0 | uint256 | The amount of token0 being sent in |
| amount1 | uint256 | The amount of token1 being sent in |

Return Values:

| Name | Type | Description |
|-----------|---------|--|
| liquidity | uint128 | The maximum amount of liquidity received |

getAmount0ForLiquidity

```

function getAmount0ForLiquidity(
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    uint128 liquidity
) internal returns (uint256 amount0)

```

Computes the amount of token0 for a given amount of liquidity and a price range

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioAX96 | uint160 | A sqrt price representing the first tick boundary |
| sqrtRatioBX96 | uint160 | A sqrt price representing the second tick boundary |
| liquidity | uint128 | The liquidity being valued |

Return Values:

| Name | Type | Description |
|---------|---------|----------------------|
| amount0 | uint256 | The amount of token0 |

getAmount1ForLiquidity

```

function getAmount1ForLiquidity(
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    uint128 liquidity
) internal returns (uint256 amount1)

```

Computes the amount of token1 for a given amount of liquidity and a price range

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioAX96 | uint160 | A sqrt price representing the first tick boundary |
| sqrtRatioBX96 | uint160 | A sqrt price representing the second tick boundary |
| liquidity | uint128 | The liquidity being valued |

Return Values:

| Name | Type | Description |
|---------|---------|----------------------|
| amount1 | uint256 | The amount of token1 |

getAmountsForLiquidity

```
function getAmountsForLiquidity(
    uint160 sqrtRatioX96,
    uint160 sqrtRatioAX96,
    uint160 sqrtRatioBX96,
    uint128 liquidity
) internal returns (uint256 amount0, uint256 amount1)
```

Computes the token0 and token1 value for a given amount of liquidity, the current pool prices and the prices at the tick boundaries

Parameters:

| Name | Type | Description |
|---------------|---------|--|
| sqrtRatioX96 | uint160 | A sqrt price representing the current pool prices |
| sqrtRatioAX96 | uint160 | A sqrt price representing the first tick boundary |
| sqrtRatioBX96 | uint160 | A sqrt price representing the second tick boundary |
| liquidity | uint128 | The liquidity being valued |

Return Values:

| Name | Type | Description |
|---------|---------|----------------------|
| amount0 | uint256 | The amount of token0 |
| amount1 | uint256 | The amount of token1 |

Functions

constructTokenURI

```
function constructTokenURI(
) public returns (string)
```

escapeQuotes

```
function escapeQuotes(
) internal returns (string)
```

tickToDecimalString

```
function tickToDecimalString(
) internal returns (string)
```

fixedPointToDecimalString

```
function fixedPointToDecimalString(
) internal returns (string)
```

feeToPercentString

```
function feeToPercentString(
) internal returns (string)
```

addressToString

```
function addressToString()  
) internal returns (string)
```

generateSVGImage

```
function generateSVGImage()  
) internal returns (string svg)
```

tokenToColorHex

```
function tokenToColorHex()  
) internal returns (string str)
```

getCircleCoord

```
function getCircleCoord()  
) internal returns (uint256)
```

sliceTokenHex

```
function sliceTokenHex()  
) internal returns (uint256)
```

Provides a function for generating an SVG associated with a Uniswap NFT

Functions

generateSVG

```
function generateSVG()  
) internal returns (string svg)
```

getCurve

```
function getCurve()  
) internal returns (string curve)
```

generateSVGCurveCircle

```
function generateSVGCurveCircle()  
) internal returns (string svg)
```

rangeLocation

```
function rangeLocation()  
) internal returns (string, string)
```

isRare

```
function isRare()  
) internal returns (bool)
```

Provides functions to integrate with V3 pool oracle

Functions

consult

```
function consult(
    address pool,
    uint32 period
) internal view returns (int24 arithmeticMeanTick, uint128 harmonicMeanLiquidity)
```

Calculates time-weighted means of tick and liquidity for a given Uniswap V3 pool

Parameters:

| Name | Type | Description |
|--------|---------|--|
| pool | address | Address of Uniswap V3 pool that we want to observe |
| period | uint32 | Number of seconds in the past to start calculating time-weighted average |

Return Values:

| Name | Type | Description |
|-----------------------|---------|--|
| arithmeticMeanTick | int24 | The arithmetic mean tick from (block.timestamp - secondsAgo) to block.timestamp |
| harmonicMeanLiquidity | uint128 | The harmonic mean liquidity from (block.timestamp - secondsAgo) to block.timestamp |

getQuoteAtTick

```
function getQuoteAtTick(
    int24 tick,
    uint128 baseAmount,
    address baseToken,
    address quoteToken
) internal pure returns (uint256 quoteAmount)
```

Given a tick and a token amount, calculates the amount of token received in exchange

Parameters:

| Name | Type | Description |
|------------|---------|---|
| tick | int24 | Tick value used to calculate the quote |
| baseAmount | uint128 | Amount of token to be converted |
| baseToken | address | Address of an ERC20 token contract used as the baseAmount denomination |
| quoteToken | address | Address of an ERC20 token contract used as the quoteAmount denomination |

Return Values:

| Name | Type | Description |
|-------------|---------|---|
| quoteAmount | uint256 | Amount of quoteToken received for baseAmount of baseToken |

Functions

hasMultiplePools

```
function hasMultiplePools(
    bytes path
) internal returns (bool)
```

Returns true iff the path contains two or more pools

Parameters:

| Name | Type | Description |
|------|-------|-----------------------|
| path | bytes | The encoded swap path |

Return Values:

| Type | Description |
|------|---|
| bool | if path contains two or more pools, otherwise false |

decodeFirstPool

```
function decodeFirstPool(
    bytes path
) internal returns (address tokenA, address tokenB, uint24 fee)
```

Decodes the first pool in path

Parameters:

| Name | Type | Description |
|------|-------|-----------------------------|
| path | bytes | The bytes encoded swap path |

Return Values:

| Name | Type | Description |
|--------|---------|------------------------------------|
| tokenA | address | The first token of the given pool |
| tokenB | address | The second token of the given pool |
| fee | uint24 | The fee level of the pool |

getFirstPool

```
function getFirstPool(
    bytes path
) internal returns (bytes)
```

Gets the segment corresponding to the first pool in the path

Parameters:

| Name | Type | Description |
|------|-------|-----------------------------|
| path | bytes | The bytes encoded swap path |

Return Values:

| Type | Description |
|-------|--|
| bytes | segment containing all data necessary to target the first pool in the path |

skipToken

```
function skipToken(
    bytes path
) internal returns (bytes)
```

Skips a token + fee element from the buffer and returns the remainder

Parameters:

| Name | Type | Description |
|------|-------|---------------|
| path | bytes | The swap path |

Return Values:

| Type | Description |
|-------|--|
| bytes | remaining token + fee elements in the path |

Functions

getPoolKey

```
function getPoolKey(
    address tokenA,
    address tokenB,
    uint24 fee
) internal returns (struct PoolAddress.PoolKey)
```

Returns PoolKey: the ordered tokens with the matched fee levels

Parameters:

| Name | Type | Description |
|--------|---------|--------------------------------------|
| tokenA | address | The first token of a pool, unsorted |
| tokenB | address | The second token of a pool, unsorted |
| fee | uint24 | The fee level of the pool |

Return Values:

| Name | Type | Description |
|---------|---------------------|---|
| Poolkey | PoolAddress.PoolKey | The pool details with ordered token0 and token1 assignments |

computeAddress

```
function computeAddress(
    address factory,
    struct PoolAddress.PoolKey key
) internal returns (address pool)
```

Deterministically computes the pool address given the factory and PoolKey

Parameters:

| Name | Type | Description |
|---------|----------------------------|---|
| factory | address | The Uniswap V3 factory contract address |
| key | struct PoolAddress.PoolKey | The PoolKey |

Return Values:

| Name | Type | Description |
|------|---------|-------------------------------------|
| pool | address | The contract address of the V3 pool |

Functions

countInitializedTicksCrossed

```
function countInitializedTicksCrossed(
) internal view returns (uint32 initializedTicksCrossed)
```

This function counts the number of initialized ticks that would incur a gas cost between tickBefore and tickAfter. When tickBefore and/or tickAfter themselves are initialized, the logic over whether we should count them depends on the direction of the swap. If we are swapping upwards (tickAfter > tickBefore) we don't want to count tickBefore but we do want to count tickAfter. The opposite is true if we are swapping downwards.

Functions

compute

```
function compute(
```

```
) internal returns (bytes32)
```

Returns the key of the position in the core library

TokenRatioSortOrder

```
library TokenRatioSortOrder {
    int256 constant NUMERATOR_MOST = 300;
    int256 constant NUMERATOR_MORE = 200;
    int256 constant NUMERATOR = 100;
    int256 constant DENOMINATOR_MOST = -300;
    int256 constant DENOMINATOR_MORE = -200;
    int256 constant DENOMINATOR = -100;
}
```

Functions

safeTransferFrom

```
function safeTransferFrom(
    address token,
    address from,
    address to,
    uint256 value
) internal
```

Transfers tokens from the targeted address to the given destination Errors with 'STF' if transfer fails

Parameters:

| Name | Type | Description |
|-------|---------|---|
| token | address | The contract address of the token to be transferred |
| from | address | The originating address from which the tokens will be transferred |
| to | address | The destination address of the transfer |
| value | uint256 | The amount to be transferred |

safeTransfer

```
function safeTransfer(
    address token,
    address to,
    uint256 value
) internal
```

Transfers tokens from msg.sender to a recipient

Errors with ST if transfer fails

Parameters:

| Name | Type | Description |
|-------|---------|---|
| token | address | The contract address of the token which will be transferred |
| to | address | The recipient of the transfer |
| value | uint256 | The value of the transfer |

safeApprove

```
function safeApprove(
    address token,
    address to,
```

```

        uint256 value
    ) internal

```

Approves the stipulated contract to spend the given allowance in the given token

Errors with 'SA' if transfer fails

Parameters:

| Name | Type | Description |
|-------|---------|---|
| token | address | The contract address of the token to be approved |
| to | address | The target of the approval |
| value | uint256 | The amount of the given token the target will be allowed to spend |

safeTransferETH

```

function safeTransferETH(
    address to,
    uint256 value
) internal

```

Transfers ETH to the recipient address

Fails with STE

Parameters:

| Name | Type | Description |
|---|---------|---------------------------------|
| to | address | The destination of the transfer |
| value | uint256 | The value to be transferred |
| Provides functions to integrate with different tier oracles of the same V3 pair | | |

Functions

consult

```

function consult(
    address pool,
    uint32 period
) internal view returns (struct WeightedOracleLibrary.PeriodObservation observation)

```

Fetches a time-weighted observation for a given Uniswap V3 pool

Parameters:

| Name | Type | Description |
|--------|---------|--|
| pool | address | Address of the pool that we want to observe |
| period | uint32 | Number of seconds in the past to start calculating the time-weighted observation |

Return Values:

| Name | Type | Description |
|-------------|---------|---|
| observation | address | An observation that has been time-weighted from (block.timestamp - period) to block.timestamp |

getArithmeticMeanTickWeightedByLiquidity

```

function getArithmeticMeanTickWeightedByLiquidity(
    struct WeightedOracleLibrary.PeriodObservation[] observations
) internal pure returns (int24 arithmeticMeanWeightedTick)

```

Given some time-weighted observations, calculates the arithmetic mean tick, weighted by liquidity

In most scenarios, each entry of `observations` should share the same `period` and underlying `pool tokens`. If `period` differs across observations, the result becomes difficult to interpret and is likely biased/manipulable. If the underlying `pool tokens` differ across observations, extreme care must be taken to ensure that both prices and liquidity values are comparable. Even if prices are commensurate (e.g. two different USD-stable assets against ETH), liquidity values may not be, as decimals can differ between tokens.

Parameters:

| Name | Type | Description |
|---------------------------|---|--------------------------------------|
| <code>observations</code> | <code>struct WeightedOracleLibrary.PeriodObservation[]</code> | A list of time-weighted observations |

Return Values:

| Name | Type | Description |
|---|---|--|
| <code>arithmeticMeanWeightedTick</code> | <code>struct WeightedOracleLibrary.PeriodObservation[]</code> | The arithmetic mean tick, weighted by the observations' time-weighted harmonic average liquidity |

sidebar_label: Uniswap V3 Staker Design sidebar_position: 1

Uniswap V3 Staker Design

The liquidity mining staker design is comprised of one canonical position staking contract, `Staker`. The technical reference for this contract is [here](#) and the source code is [here](#).

Data Structures

```
struct Incentive {
    uint128 totalRewardUnclaimed;
    uint128 numberOfStakes;
    uint160 totalSecondsClaimedX128;
}

struct Deposit {
    address owner;
    uint96 numberOfStakes;
}

struct Stake {
    uint160 secondsPerLiquidityInsideInitialX128;
    uint128 liquidity;
}

struct IncentiveKey {
    IERC20Minimal rewardToken;
    IUniswapV3Pool pool;
    uint256 startTime;
    uint256 endTime;
    address refundee;
}
```

State:

```
IUniswapV3Factory public immutable factory;
INonfungiblePositionManager public immutable nonfungiblePositionManager;

/// @dev bytes32 refers to the return value of IncentiveId.compute
mapping(bytes32 => Incentive) public incentives;

/// @dev deposits[tokenId] => Deposit
mapping(uint256 => Deposit) public deposits;

/// @dev stakes[tokenId][incentiveHash] => Stake
mapping(uint256 => mapping(bytes32 => Stake)) public stakes;

/// @dev rewards[rewardToken][msg.sender] => uint256
mapping(address => mapping(address => uint256)) public rewards;
```

Params:

```
struct CreateIncentiveParams {
    address rewardToken;
    address pool;
    uint256 startTime;
    uint256 endTime;
    uint128 totalReward;
}

struct EndIncentiveParams {
    address creator;
    address rewardToken;
    address pool;
    uint256 startTime;
    uint256 endTime;
}
```

Incentives

`createIncentive(CreateIncentiveParams memory params)`

`createIncentive` creates a liquidity mining incentive program. The key used to look up an Incentive is the hash of its immutable properties.

Check:

- Incentive with these params does not already exist
- Timestamps: `params.endTime >= params.startTime`, `params.startTime >= block.timestamp`
- Incentive with this ID does not already exist.

Effects:

- Sets `incentives[key] = Incentive(totalRewardUnclaimed=totalReward, totalSecondsClaimedX128=0, rewardToken=rewardToken)`

Interaction:

- Transfers `params.totalReward` from `msg.sender` to `self`.
 - Make sure there is a check here and it fails if the transfer fails
- Emits `IncentiveCreated`

`endIncentive(EndIncentiveParams memory params)`

`endIncentive` can be called by anyone to end an Incentive after the `endTime` has passed, transferring `totalRewardUnclaimed` of `rewardToken` back to `refundee`.

Check:

- `block.timestamp > params.endTime`
- Incentive exists (`incentive.totalRewardUnclaimed != 0`)

Effects:

- deletes `incentives[key]` (This is a new change)

Interactions:

- `safeTransfers totalRewardUnclaimed of rewardToken to the incentive creator msg.sender`
- emits `IncentiveEnded`

Deposit/Withdraw Token

Interactions

- `nonfungiblePositionManager.safeTransferFrom(sender, this, tokenId)`
 - This transfer triggers the `onERC721Received` hook

`onERC721Received(address, address from, uint256 tokenId, bytes calldata data)`

Check:

- Make sure sender is univ3 nft

Effects:

- Creates a deposit for the token setting `deposit.owner` to `from`.
 - Setting `owner` to `from` ensures that the owner of the token also owns the deposit. Approved addresses and operators may first transfer the token to themselves before depositing for deposit ownership.
- If `data.length>0`, stakes the token in one or more incentives

```
withdrawToken(uint256 tokenId, address to, bytes memory data)
```

Checks

- Check that a Deposit exists for the token and that `msg.sender` is the `owner` on that Deposit.
- Check that `numberOfStakes` on that Deposit is 0.

Effects

- Delete the Deposit `delete deposits[tokenId]`.

Interactions

- `safeTransferFrom` the token to `to` with `data`.
- `emit DepositTransferred(token, deposit.owner, address(0))`

Stake/Unstake/Rewards

stakeToken

Check:

- `deposits[params.tokenId].owner == msg.sender`
- Make sure incentive actually exists and has reward that could be claimed (`incentive.rewardToken != address(0)`)
 - Check if this check can check `totalRewardUnclaimed` instead
- Make sure token not already staked

claimReward

Interactions

- `msg.sender` to withdraw all of their reward balance in a specific token to a specified `to` address.
- `emit RewardClaimed(to, reward)`

unstakeToken

To unstake an NFT, you call `unstakeToken`, which takes all the same arguments as `stake`.

Checks

- It checks that you are the owner of the Deposit
- It checks that there exists a `Stake` for the provided key (with `exists=true`).

Effects

- Deletes the Stake.
- Decrement `numberOfStakes` for the Deposit by 1.
- `totalRewardsUnclaimed` is decremented by `reward`.
- `totalSecondsClaimed` is incremented by `seconds`.
- Increments `rewards[rewardToken][msg.sender]` by the amount given by `getRewardInfo`.

getRewardInfo

- It computes `secondsInsideX128` (the total liquidity seconds for which rewards are owed) for a given Stake, by:
 - using `snapshotCumulativesInside` from the Uniswap v3 core contract to get `secondsPerLiquidityInRangeX128`, and subtracting `secondsPerLiquidityInRangeInitialX128`.
 - Multiplying that by `stake.liquidity` to get the total seconds accrued by the liquidity in that period
- Note that X128 means it's a `UQ32X128`.
- It computes `totalSecondsUnclaimed` by taking `max(endTime, block.timestamp) - startTime`, casting it as a Q128, and subtracting `totalSecondsClaimedX128`.
- It computes `rewardRate` for the Incentive casting `incentive.totalRewardUnclaimed` as a Q128, then dividing it by `totalSecondsUnclaimedX128`.
- `reward` is then calculated as `secondsInsideX128` times the `rewardRate`, scaled down to a regular `uint128`.

sidebar_label: Uniswap V3 Staker Contract sidebar_position: 2

Uniswap V3 Staker Contract

Below is the technical reference for the staker contract, [UniswapV3Staker.sol](#). A technical guide for interacting with this staking contract will be released soon.

Functions

stakes

```
function stakes(
    uint256 tokenId,
    bytes32 incentiveId
) public view override returns (uint160 secondsPerLiquidityInsideInitialX128, uint128 liquidity)
```

Returns information about a staked liquidity NFT

Parameters:

| Name | Type | Description |
|-------------|---------|---|
| tokenId | uint256 | The ID of the staked token |
| incentiveId | bytes32 | The ID of the incentive for which the token is staked |

Return Values:

| Name | Type | Description |
|--------------------------------------|---------|--|
| secondsPerLiquidityInsideInitialX128 | uint160 | secondsPerLiquidity represented as a UQ32.128 |
| liquidity | bytes32 | The amount of liquidity in the NFT as of the last time the rewards were computed |

constructor

```
function constructor(
    contract IUniswapV3Factory _factory,
    contract INonfungiblePositionManager _nonfungiblePositionManager,
    uint256 _maxIncentiveStartLeadTime,
    uint256 _maxIncentiveDuration
) public
```

Parameters:

| Name | Type | Description |
|-----------------------------|--------------------------------------|--|
| _factory | contract IUniswapV3Factory | the Uniswap V3 factory |
| _nonfungiblePositionManager | contract INonfungiblePositionManager | the NFT position manager contract address |
| _maxIncentiveStartLeadTime | uint256 | the max duration of an incentive in seconds |
| _maxIncentiveDuration | uint256 | the max amount of seconds into the future the incentive startTime can be set |

createIncentive

```
function createIncentive(
    struct IUniswapV3Staker.IncentiveKey key,
    uint256 reward
) external
```

Creates a new liquidity mining incentive program

Parameters:

| Name | Type | Description |
|--------|--------------------------------------|---|
| key | struct IUniswapV3Staker.IncentiveKey | Details of the incentive to create |
| reward | uint256 | The amount of reward tokens to be distributed |

endIncentive

```
function endIncentive(
    struct IUniswapV3Staker.IncentiveKey key
) external returns (uint256 refund)
```

Ends an incentive after the incentive end time has passed and all stakes have been withdrawn

Parameters:

| Name | Type | Description |
|------|--------------------------------------|---------------------------------|
| key | struct IUniswapV3Staker.IncentiveKey | Details of the incentive to end |

Return Values:

| Name | Type | Description |
|--------|---------|---|
| refund | uint256 | The remaining reward tokens when the incentive is ended |

onERC721Received

```
function onERC721Received(
) external returns (bytes4)
```

Upon receiving a Uniswap V3 ERC721, creates the token deposit setting owner to `from`. Also stakes token in one or more incentives if properly formatted `data` has a length > 0.

Whenever an `{IERC721}` `tokenId` token is transferred to this contract via `{IERC721-safeTransferFrom}` by `operator` from `from`, this function is called. It must return its Solidity selector to confirm the token transfer. If any other value is returned or the interface is not implemented by the recipient, the transfer will be reverted. The selector can be obtained in Solidity with `IERC721.onERC721Received.selector`.

transferDeposit

```
function transferDeposit(
    uint256 tokenId,
    address to
) external
```

Transfers ownership of a deposit from the sender to the given recipient

Parameters:

| Name | Type | Description |
|---------|---------|---|
| tokenId | uint256 | The ID of the token (and the deposit) to transfer |
| to | address | The new owner of the deposit |

withdrawToken

```
function withdrawToken(
    uint256 tokenId,
    address to,
    bytes data
) external
```

Withdraws a Uniswap V3 LP token `tokenId` from this contract to the recipient `to`

Parameters:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|---------|---------|---|
| tokenId | uint256 | The unique identifier of an Uniswap V3 LP token |
| to | address | The address where the LP token will be sent |
| data | bytes | An optional data array that will be passed along to the <code>to</code> address via the NFT <code>safeTransferFrom</code> |

stakeToken

```
function stakeToken(
    struct IUniswapV3Staker.IncentiveKey key,
    uint256 tokenId
) external
```

Stakes a Uniswap V3 LP token

Parameters:

| Name | Type | Description |
|---------|--------------------------------------|---|
| key | struct IUniswapV3Staker.IncentiveKey | The key of the incentive for which to stake the NFT |
| tokenId | uint256 | The ID of the token to stake |

unstakeToken

```
function unstakeToken(
    struct IUniswapV3Staker.IncentiveKey key,
    uint256 tokenId
) external
```

Unstakes a Uniswap V3 LP token

Parameters:

| Name | Type | Description |
|---------|--------------------------------------|---|
| key | struct IUniswapV3Staker.IncentiveKey | The key of the incentive for which to unstake the NFT |
| tokenId | uint256 | The ID of the token to unstake |

claimReward

```
function claimReward(
    contract IERC20Minimal rewardToken,
    address to,
    uint256 amountRequested
) external override returns (uint256 reward)
```

Transfers `amountRequested` of accrued `rewardToken` rewards from the contract to the recipient `to`

Parameters:

| Name | Type | Description |
|-----------------|------------------------|--|
| rewardToken | contract IERC20Minimal | The token being distributed as a reward |
| to | address | The address where claimed rewards will be sent to |
| amountRequested | uint256 | The amount of reward tokens to claim. Claims entire reward amount if set to 0. |

Return Values:

| Name | Type | Description |
|--------|---------|-------------------------------------|
| reward | uint256 | The amount of reward tokens claimed |

getRewardInfo

```

function getRewardInfo(
    struct IUniswapV3Staker.IncentiveKey key,
    uint256 tokenId
) external view override returns (uint256 reward, uint160 secondsInsideX128)

```

Calculates the reward amount that will be received for the given stake

Parameters:

| Name | Type | Description |
|---------|--------------------------------------|--------------------------|
| key | struct IUniswapV3Staker.IncentiveKey | The key of the incentive |
| tokenId | uint256 | The ID of the token |

Return Values:

| Name | Type | Description |
|---|---------|--|
| reward | uint256 | The reward accrued to the NFT for the given incentive thus far |
| secondsInsideX128 | uint160 | The seconds inside the tick range |
| Allows staking nonfungible liquidity tokens in exchange for reward tokens | | |

Functions

factory

```

function factory()
) external view returns (contract IUniswapV3Factory)

```

The Uniswap V3 Factory

nonfungiblePositionManager

```

function nonfungiblePositionManager(
) external view returns (contract INonfungiblePositionManager)

```

The nonfungible position manager with which this staking contract is compatible

maxIncentiveDuration

```

function maxIncentiveDuration(
) external view returns (uint256)

```

The max duration of an incentive in seconds

maxIncentiveStartLeadTime

```

function maxIncentiveStartLeadTime(
) external view returns (uint256)

```

The max amount of seconds into the future the incentive startTime can be set

incentives

```

function incentives(
    bytes32 incentiveId
) external view returns (uint256 totalRewardUnclaimed, uint160 totalSecondsClaimedX128, uint96 numberOfStakes)

```

Represents a staking incentive

Parameters:

| Name | Type | Description |
|-------------|---------|--|
| incentiveId | bytes32 | The ID of the incentive computed from its parameters |

Return Values:

| Name | Type | Description |
|-------------------------|---------|---|
| totalRewardUnclaimed | uint256 | The amount of reward token not yet claimed by users |
| totalSecondsClaimedX128 | uint160 | Total liquidity-seconds claimed, represented as a UQ32.128 |
| numberOfStakes | uint96 | The count of deposits that are currently staked for the incentive |

deposits

```
function deposits()
) external view returns (address owner, uint48 numberOfStakes, int24 tickLower, int24 tickUpper)
```

Returns information about a deposited NFT

Return Values:

| Name | Type | Description |
|----------------|---------|--|
| owner | address | The owner of the deposited NFT |
| numberOfStakes | uint48 | Counter of how many incentives for which the liquidity is staked |
| tickLower | int24 | The lower tick of the range |
| tickUpper | int24 | The upper tick of the range |

stakes

```
function stakes(
    uint256 tokenId,
    bytes32 incentiveId
) external view returns (uint160 secondsPerLiquidityInsideInitialX128, uint128 liquidity)
```

Returns information about a staked liquidity NFT

Parameters:

| Name | Type | Description |
|-------------|---------|---|
| tokenId | uint256 | The ID of the staked token |
| incentiveId | bytes32 | The ID of the incentive for which the token is staked |

Return Values:

| Name | Type | Description |
|--------------------------------------|---------|--|
| secondsPerLiquidityInsideInitialX128 | uint160 | secondsPerLiquidity represented as a UQ32.128 |
| liquidity | uint128 | The amount of liquidity in the NFT as of the last time the rewards were computed |

rewards

```
function rewards(
    contract IERC20Minimal rewardToken,
    address owner
) external view returns (uint256 rewardsOwed)
```

Returns amounts of reward tokens owed to a given address according to the last time all stakes were updated

Parameters:

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|-------------|------------------------|--|
| rewardToken | contract IERC20Minimal | The token for which to check rewards |
| owner | address | The owner for which the rewards owed are checked |

Return Values:

| Name | Type | Description |
|-------------|---------|---|
| rewardsOwed | uint256 | The amount of the reward token claimable by the owner |

createIncentive

```
function createIncentive(
    struct IUniswapV3Staker.IncentiveKey key,
    uint256 reward
) external
```

Creates a new liquidity mining incentive program

Parameters:

| Name | Type | Description |
|--------|--------------------------------------|---|
| key | struct IUniswapV3Staker.IncentiveKey | Details of the incentive to create |
| reward | uint256 | The amount of reward tokens to be distributed |

endIncentive

```
function endIncentive(
    struct IUniswapV3Staker.IncentiveKey key
) external returns (uint256 refund)
```

Ends an incentive after the incentive end time has passed and all stakes have been withdrawn

Parameters:

| Name | Type | Description |
|------|--------------------------------------|---------------------------------|
| key | struct IUniswapV3Staker.IncentiveKey | Details of the incentive to end |

Return Values:

| Name | Type | Description |
|--------|---------|---|
| refund | uint256 | The remaining reward tokens when the incentive is ended |

transferDeposit

```
function transferDeposit(
    uint256 tokenId,
    address to
) external
```

Transfers ownership of a deposit from the sender to the given recipient

Parameters:

| Name | Type | Description |
|---------|---------|---|
| tokenId | uint256 | The ID of the token (and the deposit) to transfer |
| to | address | The new owner of the deposit |

withdrawToken

```
function withdrawToken(
    uint256 tokenId,
    address to,
```

```
    bytes data  
) external
```

Withdraws a Uniswap V3 LP token `tokenId` from this contract to the recipient `to`

Parameters:

| Name | Type | Description |
|----------------------|----------------------|---|
| <code>tokenId</code> | <code>uint256</code> | The unique identifier of an Uniswap V3 LP token |
| <code>to</code> | <code>address</code> | The address where the LP token will be sent |
| <code>data</code> | <code>bytes</code> | An optional data array that will be passed along to the <code>to</code> address via the NFT <code>safeTransferFrom</code> |

stakeToken

```
function stakeToken(  
    struct IUniswapV3Staker.IncentiveKey key,  
    uint256 tokenId  
) external
```

Stakes a Uniswap V3 LP token

Parameters:

| Name | Type | Description |
|----------------------|---|---|
| <code>key</code> | struct <code>IUniswapV3Staker.IncentiveKey</code> | The key of the incentive for which to stake the NFT |
| <code>tokenId</code> | <code>uint256</code> | The ID of the token to stake |

unstakeToken

```
function unstakeToken(  
    struct IUniswapV3Staker.IncentiveKey key,  
    uint256 tokenId  
) external
```

Unstakes a Uniswap V3 LP token

Parameters:

| Name | Type | Description |
|----------------------|---|---|
| <code>key</code> | struct <code>IUniswapV3Staker.IncentiveKey</code> | The key of the incentive for which to unstake the NFT |
| <code>tokenId</code> | <code>uint256</code> | The ID of the token to unstake |

claimReward

```
function claimReward(  
    contract IERC20Minimal rewardToken,  
    address to,  
    uint256 amountRequested  
) external returns (uint256 reward)
```

Transfers `amountRequested` of accrued `rewardToken` rewards from the contract to the recipient `to`

Parameters:

| Name | Type | Description |
|------------------------------|-------------------------------------|--|
| <code>rewardToken</code> | contract <code>IERC20Minimal</code> | The token being distributed as a reward |
| <code>to</code> | <code>address</code> | The address where claimed rewards will be sent to |
| <code>amountRequested</code> | <code>uint256</code> | The amount of reward tokens to claim. Claims entire reward amount if set to 0. |

Return Values:

| Name | Type | Description |
|--------|---------|-------------------------------------|
| reward | uint256 | The amount of reward tokens claimed |

getRewardInfo

```
function getRewardInfo(
    struct IUniswapV3Staker.IncentiveKey key,
    uint256 tokenId
) external returns (uint256 reward, uint160 secondsInsideX128)
```

Calculates the reward amount that will be received for the given stake

Parameters:

| Name | Type | Description |
|---------|--------------------------------------|--------------------------|
| key | struct IUniswapV3Staker.IncentiveKey | The key of the incentive |
| tokenId | uint256 | The ID of the token |

Return Values:

| Name | Type | Description |
|-------------------|---------|--|
| reward | uint256 | The reward accrued to the NFT for the given incentive thus far |
| secondsInsideX128 | uint160 | The seconds inside the tick range |

Events

IncentiveCreated

```
event IncentiveCreated(
    contract IERC20Minimal rewardToken,
    contract IUniswapV3Pool pool,
    uint256 startTime,
    uint256 endTime,
    address refundee,
    uint256 reward
)
```

Event emitted when a liquidity mining incentive has been created

Parameters:

| Name | Type | Description |
|-------------|-------------------------|---|
| rewardToken | contract IERC20Minimal | The token being distributed as a reward |
| pool | contract IUniswapV3Pool | The Uniswap V3 pool |
| startTime | uint256 | The time when the incentive program begins |
| endTime | uint256 | The time when rewards stop accruing |
| refundee | address | The address which receives any remaining reward tokens after the end time |
| reward | uint256 | The amount of reward tokens to be distributed |

IncentiveEnded

```
event IncentiveEnded(
    bytes32 incentiveId,
    uint256 refund
)
```

Event that can be emitted when a liquidity mining incentive has ended

Parameters:

| Name | Type | Description |
|-------------|---------|--------------------------------------|
| incentiveId | bytes32 | The incentive which is ending |
| refund | uint256 | The amount of reward tokens refunded |

DepositTransferred

```
event DepositTransferred(
    uint256 tokenId,
    address oldOwner,
    address newOwner
)
```

Emitted when ownership of a deposit changes

Parameters:

| Name | Type | Description |
|----------|---------|---|
| tokenId | uint256 | The ID of the deposit (and token) that is being transferred |
| oldOwner | address | The owner before the deposit was transferred |
| newOwner | address | The owner after the deposit was transferred |

TokenStaked

```
event TokenStaked(
    uint256 tokenId,
    bytes32 liquidity,
    uint128 incentiveId
)
```

Event emitted when a Uniswap V3 LP token has been staked

Parameters:

| Name | Type | Description |
|-------------|---------|---|
| tokenId | uint256 | The unique identifier of an Uniswap V3 LP token |
| liquidity | bytes32 | The amount of liquidity staked |
| incentiveId | uint128 | The incentive in which the token is staking |

TokenUnstaked

```
event TokenUnstaked(
    uint256 tokenId,
    bytes32 incentiveId
)
```

Event emitted when a Uniswap V3 LP token has been unstaked

Parameters:

| Name | Type | Description |
|-------------|---------|---|
| tokenId | uint256 | The unique identifier of an Uniswap V3 LP token |
| incentiveId | bytes32 | The incentive in which the token is staking |

RewardClaimed

```
event RewardClaimed(
    address to,
```

```

        uint256 reward
    )

```

Event emitted when a reward token has been claimed

Parameters:

| Name | Type | Description |
|--------|---------|--|
| to | address | The address where claimed rewards were sent to |
| reward | uint256 | The amount of reward tokens claimed |

Functions

compute

```

function compute(
    struct IUniswapV3Staker.IncentiveKey key
) internal pure returns (bytes32 incentiveId)

```

Calculate the key for a staking incentive

Parameters:

| Name | Type | Description |
|------|--------------------------------------|---|
| key | struct IUniswapV3Staker.IncentiveKey | The components used to compute the incentive identifier |

Return Values:

| Name | Type | Description |
|--|---------|----------------------------------|
| incentiveId | bytes32 | The identifier for the incentive |
| Encapsulates the logic for getting info about a NFT token ID | | |

Functions

getPositionInfo

```

function getPositionInfo(
    contract IUniswapV3Factory factory,
    contract INonfungiblePositionManager nonfungiblePositionManager,
    uint256 tokenId
) internal view returns (contract IUniswapV3Pool pool, int24 tickLower, int24 tickUpper, uint128 liquidity)

```

Parameters:

| Name | Type | Description |
|----------------------------|--------------------------------------|--|
| factory | contract IUniswapV3Factory | The address of the Uniswap V3 Factory used in computing the pool address |
| nonfungiblePositionManager | contract INonfungiblePositionManager | The address of the nonfungible position manager to query |
| tokenId | uint256 | The unique identifier of an Uniswap V3 LP token |

Return Values:

| Name | Type | Description |
|-----------|----------------|---|
| pool | IUniswapV3Pool | The address of the Uniswap V3 pool |
| tickLower | int24 | The lower tick of the Uniswap V3 position |
| tickUpper | int24 | The upper tick of the Uniswap V3 position |
| liquidity | uint128 | The amount of liquidity staked |

Allows computing rewards given some parameters of stakes and incentives

Functions

computeRewardAmount

```
function computeRewardAmount(
    uint256 totalRewardUnclaimed,
    uint160 totalSecondsClaimedX128,
    uint256 startTime,
    uint256 endTime,
    uint128 liquidity,
    uint160 secondsPerLiquidityInsideInitialX128,
    uint160 secondsPerLiquidityInsideX128,
    uint256 currentTime
) internal pure returns (uint256 reward, uint160 secondsInsideX128)
```

Compute the amount of rewards owed given parameters of the incentive and stake

Parameters:

| Name | Type | Description |
|--------------------------------------|---------|---|
| totalRewardUnclaimed | uint256 | The total amount of unclaimed rewards left for an incentive |
| totalSecondsClaimedX128 | uint160 | How many full liquidity-seconds have been already claimed for the incentive |
| startTime | uint256 | When the incentive rewards began in epoch seconds |
| endTime | uint256 | When rewards are no longer being dripped out in epoch seconds |
| liquidity | uint128 | The amount of liquidity, assumed to be constant over the period over which the snapshots are measured |
| secondsPerLiquidityInsideInitialX128 | uint160 | The seconds per liquidity of the liquidity tick range as of the beginning of the period |
| secondsPerLiquidityInsideX128 | uint160 | The seconds per liquidity of the liquidity tick range as of the current block timestamp |
| currentTime | uint256 | The current block timestamp, which must be greater than or equal to the start time |

Return Values:

| Name | Type | Description |
|-------------------|---------|---|
| reward | uint256 | The amount of rewards owed |
| secondsInsideX128 | uint160 | The total liquidity seconds inside the position's range for the duration of the stake |

Functions

encode

```
function encode(
) external returns (string)
```

getGasCostOfEncode

```
function getGasCostOfEncode(
) external returns (uint256)
```

Functions

getLiquidityForAmount0

```
function getLiquidityForAmount0(
) external returns (uint128 liquidity)
```

getGasCostOfGetLiquidityForAmount0

```
function getGasCostOfGetLiquidityForAmount0(
) external returns (uint256)
```

getLiquidityForAmount1

```
function getLiquidityForAmount1(
) external returns (uint128 liquidity)
```

getGasCostOfGetLiquidityForAmount1

```
function getGasCostOfGetLiquidityForAmount1(
) external returns (uint256)
```

getLiquidityForAmounts

```
function getLiquidityForAmounts(
) external returns (uint128 liquidity)
```

getGasCostOfGetLiquidityForAmounts

```
function getGasCostOfGetLiquidityForAmounts(
) external returns (uint256)
```

getAmount0ForLiquidity

```
function getAmount0ForLiquidity(
) external returns (uint256 amount0)
```

getGasCostOfGetAmount0ForLiquidity

```
function getGasCostOfGetAmount0ForLiquidity(
) external returns (uint256)
```

getAmount1ForLiquidity

```
function getAmount1ForLiquidity(
) external returns (uint256 amount1)
```

getGasCostOfGetAmount1ForLiquidity

```
function getGasCostOfGetAmount1ForLiquidity(
) external returns (uint256)
```

getAmountsForLiquidity

```
function getAmountsForLiquidity(
) external returns (uint256 amount0, uint256 amount1)
```

getGasCostOfGetAmountsForLiquidity

```
function getGasCostOfGetAmountsForLiquidity(
) external returns (uint256)
```

Functions

constructor

```
function constructor()  
) public
```

_blockTimestamp

```
function _blockTimestamp()  
) internal returns (uint256)
```

setTime

```
function setTime()  
) external
```

Functions

constructor

```
function constructor()  
) public
```

_blockTimestamp

```
function _blockTimestamp()  
) internal returns (uint256)
```

setTime

```
function setTime()  
) external
```

Functions

constructTokenURI

```
function constructTokenURI()  
) public returns (string)
```

getGasCostOfConstructTokenURI

```
function getGasCostOfConstructTokenURI()  
) public returns (uint256)
```

tickToDecimalString

```
function tickToDecimalString()  
) public returns (string)
```

fixedPointToDecimalString

```
function fixedPointToDecimalString()  
) public returns (string)
```

feeToPercentString

```
function feeToPercentString(
) public returns (string)
```

addressToString

```
function addressToString(
) public returns (string)
```

generateSVGImage

```
function generateSVGImage(
) public returns (string)
```

tokenToColorHex

```
function tokenToColorHex(
) public returns (string)
```

sliceTokenHex

```
function sliceTokenHex(
) public returns (uint256)
```

rangeLocation

```
function rangeLocation(
) public returns (string, string)
```

isRare

```
function isRare(
) public returns (bool)
```

Functions

hasMultiplePools

```
function hasMultiplePools(
) public returns (bool)
```

decodeFirstPool

```
function decodeFirstPool(
) public returns (address tokenA, address tokenB, uint24 fee)
```

getFirstPool

```
function getFirstPool(
) public returns (bytes)
```

skipToken

```
function skipToken(
) public returns (bytes)
```

getGasCostOfDecodeFirstPool

```
function getGasCostOfDecodeFirstPool()
) public returns (uint256)
```

Functions

constructor

```
function constructor(
) public
```

Functions

POOL_INIT_CODE_HASH

```
function POOL_INIT_CODE_HASH(
) external returns (bytes32)
```

computeAddress

```
function computeAddress(
) external returns (address)
```

getGasCostOfComputeAddress

```
function getGasCostOfComputeAddress(
) external returns (uint256)
```

Same as SelfPermit but not abstract

Functions

verifyCallback

```
function verifyCallback(
) external returns (contract IUniswapV3Pool pool)
```

Functions

constructor

```
function constructor(
) public
```

Functions

constructor

```
function constructor(
) public
```

Functions

constructor

```
function constructor(
) public
```

permit

```
function permit(
) external
```

Functions

functionThatRevertsWithError

```
function functionThatRevertsWithError(
) external
```

functionThatReturnsTuple

```
function functionThatReturnsTuple(
) external returns (struct TestMulticall.Tuple tuple)
```

pays

```
function pays(
) external
```

returnSender

```
function returnSender(
) external returns (address)
```

Functions

setOwner

```
function setOwner(
) external
```

isValidSignature

```
function isValidSignature(
) external returns (bytes4 magicValue)
```

Functions

swapExact0For1

```
function swapExact0For1(
) external
```

swap0ForExact1

```
function swap0ForExact1(
) external
```

swapExact1For0

```
function swapExact1For0(
) external
```

swap1ForExact0

```
function swap1ForExact0()  
) external
```

uniswapV3SwapCallback

```
function uniswapV3SwapCallback()  
) external
```

Functions

getGasCostOfGetPopulatedTicksInWord

```
function getGasCostOfGetPopulatedTicksInWord()  
) external returns (uint256)
```

id: overview sidebar_position: 1 title: Overview

The Uniswap Core SDK

Welcome to the V3 Core SDK!

The Uniswap Core SDK provides abstractions for other SDKs to use in a Typescript/Javascript environment. It is used throughout the Uniswap SDKs, such as the [V3 SDK](#).

This SDK is not expected to be used in isolation, but only as part of other SDKs.

We recommend taking a look at the [Technical Reference](#).

Resources

- [SDK Core Github Repo](#)
- [Core SDK NPM Package](#)

 Unit Tests  passing  Lint  passing npm@latest v4.0.2 minzipped size 40.3 kB discord join chat

[@uniswap/sdk-core](#) / [Exports](#) / CurrencyAmount

Class: CurrencyAmount<T>

Type parameters

| Name | Type |
|------|----------------------------------|
| T | extends Currency |

Hierarchy

- [Fraction](#)
└→ [CurrencyAmount](#)

Table of contents

Constructors

- [constructor](#)

Properties

- [currency](#)
- [decimalScale](#)
- [denominator](#)
- [numerator](#)

Accessors

- [asFraction](#)
- [quotient](#)
- [remainder](#)
- [wrapped](#)

Methods

- [add](#)
- [divide](#)
- [equalTo](#)
- [greaterThan](#)
- [invert](#)
- [lessThan](#)
- [multiply](#)
- [subtract](#)
- [toExact](#)
- [toFixed](#)
- [toSignificant](#)
- [fromFractionalAmount](#)
- [fromRawAmount](#)

Constructors

constructor

- Protected `new CurrencyAmount< T >(currency , numerator , denominator?)`

Type parameters

| Name | Type |
|------|----------------------------------|
| T | extends Currency |

Parameters

| Name | Type |
|--------------|---------------------------|
| currency | T |
| numerator | Bigintish |
| denominator? | Bigintish |

Overrides

[Fraction.constructor](#)

Defined in

[entities/fractions/currencyAmount.ts:40](#)

Properties

currency

- Readonly `currency: T`

Defined in

[entities/fractions/currencyAmount.ts:14](#)

decimalScale

- Readonly `decimalScale: default`

Defined in

[entities/fractions/currencyAmount.ts:15](#)

denominator

- Readonly `denominator: default`

Inherited from

[Fraction.denominator](#)

Defined in

[entities/fractions/fraction.ts:26](#)

numerator

- `Readonly numerator: default`

Inherited from

[Fraction.numerator](#)

Defined in

[entities/fractions/fraction.ts:25](#)

Accessors

asFraction

- `get asFraction(): Fraction`

Helper method for converting any super class back to a fraction

Returns

[Fraction](#)

Inherited from

`Fraction.asFraction`

Defined in

[entities/fractions/fraction.ts:154](#)

quotient

- `get quotient(): default`

Returns

`default`

Inherited from

`Fraction.quotient`

Defined in

[entities/fractions/fraction.ts:42](#)

remainder

- `get remainder(): Fraction`

Returns

[Fraction](#)

Inherited from

`Fraction.remainder`

Defined in

[entities/fractions/fraction.ts:47](#)

wrapped

- `get wrapped(): CurrencyAmount < Token >`

Returns

[CurrencyAmount](#) < [Token](#) >

Defined in

[entities/fractions/currencyAmount.ts:91](#)

Methods

add

- **add(other): [CurrencyAmount](#) < T >**

Parameters

| Name | Type |
|-------|------------------------------------|
| other | CurrencyAmount <T> |

Returns

[CurrencyAmount](#) < T >

Overrides

[Fraction.add](#)

Defined in

[entities/fractions/currencyAmount.ts:47](#)

divide

- **divide(other): [CurrencyAmount](#) < T >**

Parameters

| Name | Type |
|-------|--|
| other | Bigintish Fraction |

Returns

[CurrencyAmount](#) < T >

Overrides

[Fraction.divide](#)

Defined in

[entities/fractions/currencyAmount.ts:64](#)

equalTo

- **equalTo(other): boolean**

Parameters

| Name | Type |
|-------|--|
| other | Bigintish Fraction |

Returns

boolean

Inherited from

[Fraction.equalTo](#)

Defined in

[entities/fractions/fraction.ts:91](#)

greaterThan

- **greaterThan(other): boolean**

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

`boolean`

Inherited from

[Fraction.greaterThan](#)

Defined in

[entities/fractions/fraction.ts:99](#)

invert

- `invert(): Fraction`

Returns

[Fraction](#)

Inherited from

[Fraction.invert](#)

Defined in

[entities/fractions/fraction.ts:51](#)

lessThan

- `lessThan(other): boolean`

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

`boolean`

Inherited from

[Fraction.lessThan](#)

Defined in

[entities/fractions/fraction.ts:83](#)

multiply

- `multiply(other): CurrencyAmount < T >`

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

[CurrencyAmount](#) < T >

Overrides

[Fraction.multiply](#)

Defined in

[entities/fractions/currencyAmount.ts:59](#)

subtract

▸ **subtract**(other): [CurrencyAmount](#) < T >

Parameters

| Name | Type |
|-------|------------------------------------|
| other | CurrencyAmount <T> |

Returns

[CurrencyAmount](#) < T >

Overrides

[Fraction.subtract](#)

Defined in

[entities/fractions/currencyAmount.ts:53](#)

toExact

▸ **toExact**(format?): string

Parameters

| Name | Type |
|--------|--------|
| format | object |

Returns

string

Defined in

[entities/fractions/currencyAmount.ts:86](#)

toFixed

▸ **toFixed**(decimalPlaces? , format? , rounding?): string

Parameters

| Name | Type | Default value |
|---------------|--------------------------|---------------------|
| decimalPlaces | number | undefined |
| format? | object | undefined |
| rounding | Rounding | Rounding.ROUND_DOWN |

Returns

string

Overrides

[Fraction.toFixed](#)

Defined in

[entities/fractions/currencyAmount.ts:77](#)

toSignificant

▸ **toSignificant**(significantDigits? , format? , rounding?): string

Parameters

| Name | Type | Default value |
|-------------------|--------------------------|---------------------|
| significantDigits | number | 6 |
| format? | object | undefined |
| rounding | Rounding | Rounding.ROUND_DOWN |

Returns`string`**Overrides**[Fraction.toSignificant](#)**Defined in**[entities/fractions/currencyAmount.ts:69](#)**fromFractionalAmount**

- Static `fromFractionalAmount< T >(currency , numerator , denominator): CurrencyAmount < T >`

Construct a currency amount with a denominator that is not equal to 1

Type parameters

| Name | Type |
|------|----------------------------------|
| T | extends Currency |

Parameters

| Name | Type | Description |
|-------------|---------------------------|--|
| currency | T | the currency |
| numerator | BigintIsh | the numerator of the fractional token amount |
| denominator | BigintIsh | the denominator of the fractional token amount |

Returns`CurrencyAmount < T >`**Defined in**[entities/fractions/currencyAmount.ts:32](#)**fromRawAmount**

- Static `fromRawAmount< T >(currency , rawAmount): CurrencyAmount < T >`

Returns a new currency amount instance from the unitless amount of token, i.e. the raw amount

Type parameters

| Name | Type |
|------|----------------------------------|
| T | extends Currency |

Parameters

| Name | Type | Description |
|-----------|---------------------------|-------------------------------|
| currency | T | the currency in the amount |
| rawAmount | BigintIsh | the raw token or ether amount |

Returns`CurrencyAmount < T >`**Defined in**[entities/fractions/currencyAmount.ts:22](#) @uniswap/sdk-core / [Exports](#) / Ether

Class: Ether

Ether is the main usage of a 'native' currency, i.e. for Ethereum mainnet and all testnets

Hierarchy

- [NativeCurrency](#)

↳ [Ether](#)

Table of contents

Constructors

- [constructor](#)

Properties

- [chainId](#)
- [decimals](#)
- [isNative](#)
- [isToken](#)
- [name](#)
- [symbol](#)
- [etherCache](#)

Accessors

- [wrapped](#)

Methods

- [equals](#)
- [onChain](#)

Constructors

constructor

- Protected `new Ether(chainId)`

Parameters

| Name | Type |
|---------|--------|
| chainId | number |

Overrides

[NativeCurrency.constructor](#)

Defined in

[entities/ether.ts:11](#)

Properties

chainId

- Readonly `chainId: number`

The chain ID on which this currency resides

Inherited from

[NativeCurrency.chainId](#)

Defined in

[entities/baseCurrency.ts:21](#)

decimals

- Readonly `decimals: number`

The decimals used in representing currency amounts

Inherited from

[NativeCurrency.decimals](#)

Defined in

[entities/baseCurrency.ts:25](#)

isNative

- `Readonly isNative: true`

Inherited from

[NativeCurrency.isNative](#)

Defined in

[entities/nativeCurrency.ts:7](#)

isToken

- `Readonly isToken: false`

Inherited from

[NativeCurrency.isToken](#)

Defined in

[entities/nativeCurrency.ts:8](#)

name

- `Optional Readonly name: string`

The name of the currency, i.e. a descriptive textual non-unique identifier

Inherited from

[NativeCurrency.name](#)

Defined in

[entities/baseCurrency.ts:33](#)

symbol

- `Optional Readonly symbol: string`

The symbol of the currency, i.e. a short textual non-unique identifier

Inherited from

[NativeCurrency.symbol](#)

Defined in

[entities/baseCurrency.ts:29](#)

_etherCache

- `Static Private _etherCache: Object = {}`

Index signature

- `[chainId: number]: Ether`

Defined in

[entities/ether.ts:21](#)

Accessors

wrapped

- `get wrapped(): Token`

Return the wrapped version of this currency that can be used with the Uniswap contracts. Currencies must implement this to be used in Uniswap

Returns

[Token](#)

Overrides

NativeCurrency.wrapped

Defined in

[entities/ether.ts:15](#)

Methods

equals

- `equals(other): boolean`

Returns whether this currency is functionally equivalent to the other currency

Parameters

| Name | Type | Description |
|-------|--------------------------|--------------------|
| other | Currency | the other currency |

Returns

`boolean`

Overrides

[NativeCurrency.equals](#)

Defined in

[entities/ether.ts:27](#)

onChain

- Static `onChain(chainId): Ether`

Parameters

| Name | Type |
|---------|---------------------|
| chainId | <code>number</code> |

Returns

[Ether](#)

Defined in

[entities/ether.ts:23 @uniswap/sdk-core / Exports](#) / Fraction

Class: Fraction

Hierarchy

- `Fraction`
 - ↳ [CurrencyAmount](#)
 - ↳ [Percent](#)
 - ↳ [Price](#)

Table of contents

Constructors

- [constructor](#)

Properties

- [denominator](#)
- [numerator](#)

Accessors

- [asFraction](#)

- [quotient](#)
- [remainder](#)

Methods

- [add](#)
- [divide](#)
- [equalTo](#)
- [greaterThan](#)
- [invert](#)
- [lessThan](#)
- [multiply](#)
- [subtract](#)
- [toFixed](#)
- [toSignificant](#)
- [tryParseFraction](#)

Constructors

constructor

- `new Fraction(numerator , denominator?)`

Parameters

| Name | Type |
|-------------|---------------------------|
| numerator | Bigintish |
| denominator | Bigintish |

Defined in

[entities/fractions/fraction.ts:28](#)

Properties

denominator

- `Readonly denominator: default`

Defined in

[entities/fractions/fraction.ts:26](#)

numerator

- `Readonly numerator: default`

Defined in

[entities/fractions/fraction.ts:25](#)

Accessors

asFraction

- `get asFraction(): Fraction`

Helper method for converting any super class back to a fraction

Returns

[Fraction](#)

Defined in

[entities/fractions/fraction.ts:154](#)

quotient

- `get quotient(): default`

Returns

`default`

Defined in

[entities/fractions/fraction.ts:42](#)

remainder

- `get remainder(): Fraction`

Returns

`Fraction`

Defined in

[entities/fractions/fraction.ts:47](#)

Methods

add

- `add(other): Fraction`

Parameters

| Name | Type |
|-------|-----------------------------------|
| other | <code>BigintIsh Fraction</code> |

Returns

`Fraction`

Defined in

[entities/fractions/fraction.ts:55](#)

divide

- `divide(other): Fraction`

Parameters

| Name | Type |
|-------|-----------------------------------|
| other | <code>BigintIsh Fraction</code> |

Returns

`Fraction`

Defined in

[entities/fractions/fraction.ts:115](#)

equalTo

- `equalTo(other): boolean`

Parameters

| Name | Type |
|-------|-----------------------------------|
| other | <code>BigintIsh Fraction</code> |

Returns

`boolean`

Defined in

[entities/fractions/fraction.ts:91](#)

greaterThan

- `greaterThan(other): boolean`

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns`boolean`**Defined in**[entities/fractions/fraction.ts:99](#)**invert**

- **invert()**: [Fraction](#)

Returns`Fraction`**Defined in**[entities/fractions/fraction.ts:51](#)**lessThan**

- **lessThan(other)**: `boolean`

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns`boolean`**Defined in**[entities/fractions/fraction.ts:83](#)**multiply**

- **multiply(other)**: [Fraction](#)

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns`Fraction`**Defined in**[entities/fractions/fraction.ts:107](#)**subtract**

- **subtract(other)**: [Fraction](#)

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns`Fraction`

Defined in

[entities/fractions/fraction.ts:69](#)

toFixed

▸ **toFixed**(decimalPlaces , format? , rounding?): string

Parameters

| Name | Type | Default value |
|---------------|--------------------------|------------------------|
| decimalPlaces | number | undefined |
| format | object | undefined |
| rounding | Rounding | Rounding.ROUND_HALF_UP |

Returns

string

Defined in

[entities/fractions/fraction.ts:138](#)

toSignificant

▸ **toSignificant**(significantDigits , format? , rounding?): string

Parameters

| Name | Type | Default value |
|-------------------|--------------------------|------------------------|
| significantDigits | number | undefined |
| format | object | undefined |
| rounding | Rounding | Rounding.ROUND_HALF_UP |

Returns

string

Defined in

[entities/fractions/fraction.ts:123](#)

tryParseFraction

▸ Static Private **tryParseFraction**(fractionish): [Fraction](#)

Parameters

| Name | Type |
|-------------|--|
| fractionish | BigintIsh Fraction |

Returns

[Fraction](#)

Defined in

[entities/fractions/fraction.ts:33](#) [@uniswap/sdk-core](#) / [Exports](#) / NativeCurrency

Class: NativeCurrency

Represents the native currency of the chain on which it resides, e.g.

Hierarchy

- [BaseCurrency](#)
 - ↳ [NativeCurrency](#)

↳ [Ether](#)

Table of contents

Constructors

- [constructor](#)

Properties

- [chainId](#)
- [decimals](#)
- [isNative](#)
- [isToken](#)
- [name](#)
- [symbol](#)

Accessors

- [wrapped](#)

Methods

- [equals](#)

Constructors

constructor

- Protected `new NativeCurrency(chainId , decimals , symbol? , name?)`

Constructs an instance of the base class `BaseCurrency`.

Parameters

| Name | Type | Description |
|----------|--------|---|
| chainId | number | the chain ID on which this currency resides |
| decimals | number | decimals of the currency |
| symbol? | string | symbol of the currency |
| name? | string | of the currency |

Inherited from

`BaseCurrency.constructor`

Defined in

[entities/baseCurrency.ts:42](#)

Properties

chainId

- Readonly `chainId: number`

The chain ID on which this currency resides

Inherited from

`BaseCurrency.chainId`

Defined in

[entities/baseCurrency.ts:21](#)

decimals

- Readonly `decimals: number`

The decimals used in representing currency amounts

Inherited from

`BaseCurrency.decimals`

Defined in

[entities/baseCurrency.ts:25](#)

isNative

- `Readonly isNative: true`

Overrides

BaseCurrency.isNative

Defined in

[entities/nativeCurrency.ts:7](#)

isToken

- `Readonly isToken: false`

Overrides

BaseCurrency.isToken

Defined in

[entities/nativeCurrency.ts:8](#)

name

- `Optional Readonly name: string`

The name of the currency, i.e. a descriptive textual non-unique identifier

Inherited from

BaseCurrency.name

Defined in

[entities/baseCurrency.ts:33](#)

symbol

- `Optional Readonly symbol: string`

The symbol of the currency, i.e. a short textual non-unique identifier

Inherited from

BaseCurrency.symbol

Defined in

[entities/baseCurrency.ts:29](#)

Accessors

wrapped

- `Abstract get wrapped(): Token`

Return the wrapped version of this currency that can be used with the Uniswap contracts. Currencies must implement this to be used in Uniswap

Returns

[Token](#)

Inherited from

BaseCurrency.wrapped

Defined in

[entities/baseCurrency.ts:62](#)

Methods

equals

- Abstract `equals(other): boolean`

Returns whether this currency is functionally equivalent to the other currency

Parameters

| Name | Type | Description |
|-------|--------------------------|--------------------|
| other | Currency | the other currency |

Returns

`boolean`

Inherited from

`BaseCurrency.equals`

Defined in

[entities/baseCurrency.ts:56 @uniswap/sdk-core / Exports](#) / Percent

Class: Percent

Hierarchy

- [Fraction](#)

↳ [Percent](#)

Table of contents

Constructors

- [constructor](#)

Properties

- [denominator](#)
- [isPercent](#)
- [numerator](#)

Accessors

- [asFraction](#)
- [quotient](#)
- [remainder](#)

Methods

- [add](#)
- [divide](#)
- [equalTo](#)
- [greaterThan](#)
- [invert](#)
- [lessThan](#)
- [multiply](#)
- [subtract](#)
- [toFixed](#)
- [toSignificant](#)

Constructors

constructor

- `new Percent(numerator , denominator?)`

Parameters

| Name | Type |
|-----------|---------------------------|
| numerator | Bigintish |

| | |
|-------------|---------------------------|
| denominator | Bigintish |
|-------------|---------------------------|

Inherited from

[Fraction.constructor](#)

Defined in

[entities/fractions/fraction.ts:28](#)

Properties

denominator

- Readonly `denominator`: default

Inherited from

[Fraction.denominator](#)

Defined in

[entities/fractions/fraction.ts:26](#)

isPercent

- Readonly `isPercent`: true

This boolean prevents a fraction from being interpreted as a Percent

Defined in

[entities/fractions/percent.ts:19](#)

numerator

- Readonly `numerator`: default

Inherited from

[Fraction.numerator](#)

Defined in

[entities/fractions/fraction.ts:25](#)

Accessors

asFraction

- get `asFraction()`: [Fraction](#)

Helper method for converting any super class back to a fraction

Returns

[Fraction](#)

Inherited from

[Fraction.asFraction](#)

Defined in

[entities/fractions/fraction.ts:154](#)

quotient

- get `quotient()`: default

Returns

default

Inherited from

[Fraction.quotient](#)

Defined in

[entities/fractions/fraction.ts:42](#)

remainder

- `get remainder(): Fraction`

Returns

[Fraction](#)

Inherited from

Fraction remainder

Defined in

[entities/fractions/fraction.ts:47](#)

Methods

add

- `add(other): Percent`

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

[Percent](#)

Overrides

[Fraction.add](#)

Defined in

[entities/fractions/percent.ts:21](#)

divide

- `divide(other): Percent`

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

[Percent](#)

Overrides

[Fraction.divide](#)

Defined in

[entities/fractions/percent.ts:33](#)

equalTo

- `equalTo(other): boolean`

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

boolean

Inherited from

[Fraction.equalTo](#)

Defined in

[entities/fractions/fraction.ts:91](#)

greaterThan

- **greaterThan(other): boolean**

Parameters

| Name | Type |
|-------|--|
| other | Bigintish Fraction |

Returns

boolean

Inherited from

[Fraction.greaterThan](#)

Defined in

[entities/fractions/fraction.ts:99](#)

invert

- **invert(): Fraction**

Returns

[Fraction](#)

Inherited from

[Fraction.invert](#)

Defined in

[entities/fractions/fraction.ts:51](#)

lessThan

- **lessThan(other): boolean**

Parameters

| Name | Type |
|-------|--|
| other | Bigintish Fraction |

Returns

boolean

Inherited from

[Fraction.lessThan](#)

Defined in

[entities/fractions/fraction.ts:83](#)

multiply

- **multiply(other): Percent**

Parameters

| Name | Type |
|------|------|
| | |

| | |
|-------|--|
| other | BigintIsh Fraction |
|-------|--|

Returns

[Percent](#)

Overrides

[Fraction.multiply](#)

Defined in

[entities/fractions/percent.ts:29](#)

subtract

▸ **subtract**(other): [Percent](#)

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

[Percent](#)

Overrides

[Fraction.subtract](#)

Defined in

[entities/fractions/percent.ts:25](#)

toFixed

▸ **toFixed**(decimalPlaces? , format? , rounding?): string

Parameters

| Name | Type | Default value |
|---------------|--------------------------|---------------|
| decimalPlaces | number | 2 |
| format? | object | undefined |
| rounding? | Rounding | undefined |

Returns

string

Overrides

[Fraction.toFixed](#)

Defined in

[entities/fractions/percent.ts:41](#)

toSignificant

▸ **toSignificant**(significantDigits? , format? , rounding?): string

Parameters

| Name | Type | Default value |
|-------------------|--------------------------|---------------|
| significantDigits | number | 5 |
| format? | object | undefined |
| rounding? | Rounding | undefined |

Returns

```
string
```

Overrides

[Fraction.toSignificant](#)

Defined in

[entities/fractions/percent.ts:37 @uniswap/sdk-core / Exports / Price](#)

Class: Price<TBase, TQuote>

Type parameters

| Name | Type |
|--------|----------------------------------|
| TBase | extends Currency |
| TQuote | extends Currency |

Hierarchy

- [Fraction](#)

↳ [Price](#)

Table of contents

Constructors

- [constructor](#)

Properties

- [baseCurrency](#)
- [denominator](#)
- [numerator](#)
- [quoteCurrency](#)
- [scalar](#)

Accessors

- [adjustedForDecimals](#)
- [asFraction](#)
- [quotient](#)
- [remainder](#)

Methods

- [add](#)
- [divide](#)
- [equalTo](#)
- [greaterThan](#)
- [invert](#)
- [lessThan](#)
- [multiply](#)
- [quote](#)
- [subtract](#)
- [toFixed](#)
- [toSignificant](#)

Constructors

constructor

- `new Price< TBase , TQuote >(... args)`

Construct a price, either with the base and quote currency amount, or the

Type parameters

| Name | Type |
|------|------|
| | |

| | |
|--------|----------------------------------|
| TBase | extends Currency |
| TQuote | extends Currency |

Parameters

| Name | Type |
|---------|---|
| ...args | [TBase, TQuote, BigintIsh , BigintIsh] [{ baseAmount: CurrencyAmount <TBase> ; quoteAmount: CurrencyAmount <TQuote> }] |

Overrides

[Fraction.constructor](#)

Defined in

[entities/fractions/price.ts:18](#)

Properties

baseCurrency

- `Readonly baseCurrency: TBase`

Defined in

[entities/fractions/price.ts:10](#)

denominator

- `Readonly denominator: default`

Inherited from

[Fraction.denominator](#)

Defined in

[entities/fractions/fraction.ts:26](#)

numerator

- `Readonly numerator: default`

Inherited from

[Fraction.numerator](#)

Defined in

[entities/fractions/fraction.ts:25](#)

quoteCurrency

- `Readonly quoteCurrency: TQuote`

Defined in

[entities/fractions/price.ts:11](#)

scalar

- `Readonly scalar: Fraction`

Defined in

[entities/fractions/price.ts:12](#)

Accessors

adjustedForDecimals

- `Private get adjustedForDecimals(): Fraction`

Get the value scaled by decimals for formatting

Returns

[Fraction](#)

Defined in

[entities/fractions/price.ts:77](#)

asFraction

- `get asFraction(): Fraction`

Helper method for converting any super class back to a fraction

Returns

[Fraction](#)

Inherited from

Fraction.asFraction

Defined in

[entities/fractions/fraction.ts:154](#)

quotient

- `get quotient(): default`

Returns

`default`

Inherited from

Fraction.quotient

Defined in

[entities/fractions/fraction.ts:42](#)

remainder

- `get remainder(): Fraction`

Returns

[Fraction](#)

Inherited from

Fraction.remainder

Defined in

[entities/fractions/fraction.ts:47](#)

Methods

add

- `add(other): Fraction`

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

[Fraction](#)

Inherited from

[Fraction.add](#)

Defined in

[entities/fractions/fraction.ts:55](#)

divide

▪ **divide(other): [Fraction](#)**

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

[Fraction](#)

Inherited from

[Fraction.divide](#)

Defined in

[entities/fractions/fraction.ts:115](#)

equalTo

▪ **equalTo(other): boolean**

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

boolean

Inherited from

[Fraction.equalTo](#)

Defined in

[entities/fractions/fraction.ts:91](#)

greaterThan

▪ **greaterThan(other): boolean**

Parameters

| Name | Type |
|-------|--|
| other | BigintIsh Fraction |

Returns

boolean

Inherited from

[Fraction.greaterThan](#)

Defined in

[entities/fractions/fraction.ts:99](#)

invert

▪ **invert(): [Price](#) < TQuote , TBase >**

Flip the price, switching the base and quote currency

Returns

[Price](#) < TQuote , TBase >

Overrides

[Fraction.invert](#)

Defined in

[entities/fractions/price.ts:49](#)

lessThan

▸ **lessThan**(other): boolean

Parameters

| Name | Type |
|-------|--|
| other | Bigintish Fraction |

Returns

boolean

Inherited from

[Fraction.lessThan](#)

Defined in

[entities/fractions/fraction.ts:83](#)

multiply

▸ **multiply**< TOtherQuote >(other): [Price](#) < TBase , TOtherQuote >

Multiply the price by another price, returning a new price. The other price must have the same base currency as this price's quote currency

Type parameters

| Name | Type |
|-------------|----------------------------------|
| TOtherQuote | extends Currency |

Parameters

| Name | Type | Description |
|-------|---|-----------------|
| other | Price <TQuote, TOtherQuote> | the other price |

Returns

[Price](#) < TBase , TOtherQuote >

Overrides

[Fraction.multiply](#)

Defined in

[entities/fractions/price.ts:57](#)

quote

▸ **quote**(currencyAmount): [CurrencyAmount](#) < TQuote >

Return the amount of quote currency corresponding to a given amount of the base currency

Parameters

| Name | Type | Description |
|----------------|--|--|
| currencyAmount | CurrencyAmount <TBase> | the amount of base currency to quote against the price |

Returns

[CurrencyAmount](#) < TQuote >

Defined in

[entities/fractions/price.ts:67](#)

subtract

▸ **subtract**(other): [Fraction](#)

Parameters

| Name | Type |
|-------|--|
| other | Bigintish Fraction |

Returns

[Fraction](#)

Inherited from

[Fraction.subtract](#)

Defined in

[entities/fractions/fraction.ts:69](#)

toFixed

▸ **toFixed**(decimalPlaces? , format? , rounding?): string

Parameters

| Name | Type | Default value |
|---------------|--------------------------|---------------|
| decimalPlaces | number | 4 |
| format? | object | undefined |
| rounding? | Rounding | undefined |

Returns

string

Overrides

[Fraction.toFixed](#)

Defined in

[entities/fractions/price.ts:85](#)

toSignificant

▸ **toSignificant**(significantDigits? , format? , rounding?): string

Parameters

| Name | Type | Default value |
|-------------------|--------------------------|---------------|
| significantDigits | number | 6 |
| format? | object | undefined |
| rounding? | Rounding | undefined |

Returns

string

Overrides

[Fraction.toSignificant](#)

Defined in

[entities/fractions/price.ts:81](#) @uniswap/sdk-core / [Exports](#) / Token

Class: Token

Represents an ERC20 token with a unique address and some metadata.

Hierarchy

- BaseCurrency
 - ↳ Token

Table of contents

Constructors

- [constructor](#)

Properties

- [address](#)
- [chainId](#)
- [decimals](#)
- [isNative](#)
- [isToken](#)
- [name](#)
- [symbol](#)

Accessors

- [wrapped](#)

Methods

- [equals](#)
- [sortsBefore](#)

Constructors

constructor

• `new Token(chainId , address , decimals , symbol? , name? , bypassChecksum?)`

Parameters

| Name | Type | Description |
|-----------------|---------|--|
| chainId | number | BaseCurrency#chainId |
| address | string | The contract address on the chain on which this token lives |
| decimals | number | BaseCurrency#decimals |
| symbol? | string | BaseCurrency#symbol |
| name? | string | BaseCurrency#name |
| bypassChecksum? | boolean | If true it only checks for length === 42, startsWith 0x and contains only hex characters |

Overrides

BaseCurrency.constructor

Defined in

[entities/token.ts:27](#)

Properties

address

• `Readonly address: string`

The contract address on the chain on which this token lives

Defined in

[entities/token.ts:16](#)

chainId

- `Readonly chainId: number`

The chain ID on which this currency resides

Inherited from

BaseCurrency.chainId

Defined in

[entities/baseCurrency.ts:21](#)

decimals

- `Readonly decimals: number`

The decimals used in representing currency amounts

Inherited from

BaseCurrency.decimals

Defined in

[entities/baseCurrency.ts:25](#)

isNative

- `Readonly isNative: false`

Overrides

BaseCurrency.isNative

Defined in

[entities/token.ts:10](#)

isToken

- `Readonly isToken: true`

Overrides

BaseCurrency.isToken

Defined in

[entities/token.ts:11](#)

name

- `Optional Readonly name: string`

The name of the currency, i.e. a descriptive textual non-unique identifier

Inherited from

BaseCurrency.name

Defined in

[entities/baseCurrency.ts:33](#)

symbol

- `Optional Readonly symbol: string`

The symbol of the currency, i.e. a short textual non-unique identifier

Inherited from

BaseCurrency.symbol

Defined in

[entities/baseCurrency.ts:29](#)

Accessors

wrapped

- `get wrapped(): Token`

Return this token, which does not need to be wrapped

Returns

[Token](#)

Overrides

BaseCurrency.wrapped

Defined in

[entities/token.ts:66](#)

Methods

equals

- `equals(other): boolean`

Returns true if the two tokens are equivalent, i.e. have the same chainId and address.

Parameters

| Name | Type | Description |
|-------|--------------------------|------------------------|
| other | Currency | other token to compare |

Returns

`boolean`

Overrides

BaseCurrency.equals

Defined in

[entities/token.ts:47](#)

sortsBefore

- `sortsBefore(other): boolean`

Returns true if the address of this token sorts before the address of the other token

Throws

if the tokens have the same address

Throws

if the tokens are on different chains

Parameters

| Name | Type | Description |
|-------|-----------------------|------------------------|
| other | Token | other token to compare |

Returns

`boolean`

Defined in

[entities/token.ts:57](#) [@uniswap/sdk-core](#) / [Exports](#) / Rounding

Enumeration: Rounding

Table of contents

Enumeration Members

- [ROUND_DOWN](#)
- [ROUND_HALF_UP](#)
- [ROUND_UP](#)

Enumeration Members

ROUND_DOWN

- ROUND_DOWN = 0

Defined in

[constants.ts:32](#)

ROUND_HALF_UP

- ROUND_HALF_UP = 1

Defined in

[constants.ts:33](#)

ROUND_UP

- ROUND_UP = 2

Defined in

[constants.ts:34](#) [@uniswap/sdk-core](#) / [Exports](#) / SupportedChainId

Enumeration: SupportedChainId

Table of contents

Enumeration Members

- [ARBITRUM_ONE](#)
- [ARBITRUM_RINKEBY](#)
- [CELO](#)
- [CELO_ALFAJORES](#)
- [GOERLI](#)
- [KOVAN](#)
- [MAINNET](#)
- [OPTIMISM](#)
- [OPTIMISM_GOERLI](#)
- [POLYGON](#)
- [POLYGON_MUMBAI](#)
- [RINKEBY](#)
- [ROPSTEN](#)

Enumeration Members

ARBITRUM_ONE

- ARBITRUM_ONE = 42161

Defined in

[constants.ts:10](#)

ARBITRUM_RINKEBY

- ARBITRUM_RINKEBY = 421611

Defined in

[constants.ts:11](#)

CELO

- **CELO** = 42220

Defined in

[constants.ts:19](#)

CELO_ALFAJORES

- **CELO_ALFAJORES** = 44787

Defined in

[constants.ts:20](#)

GOERLI

- **GOERLI** = 5

Defined in

[constants.ts:7](#)

KOVAN

- **KOVAN** = 42

Defined in

[constants.ts:8](#)

MAINNET

- **MAINNET** = 1

Defined in

[constants.ts:4](#)

OPTIMISM

- **OPTIMISM** = 10

Defined in

[constants.ts:13](#)

OPTIMISM_GOERLI

- **OPTIMISM_GOERLI** = 420

Defined in

[constants.ts:14](#)

POLYGON

- **POLYGON** = 137

Defined in

[constants.ts:16](#)

POLYGON_MUMBAI

- **POLYGON_MUMBAI** = 80001

Defined in

[constants.ts:17](#)

RINKEBY

- RINKEBY = 4

Defined in

[constants.ts:6](#)

ROPSTEN

- ROPSTEN = 3

Defined in

[constants.ts:5](#) @uniswap/sdk-core / [Exports](#) / TradeType

Enumeration: TradeType

Table of contents

Enumeration Members

- [EXACT_INPUT](#)
- [EXACT_OUTPUT](#)

Enumeration Members

EXACT_INPUT

- EXACT_INPUT = 0

Defined in

[constants.ts:27](#)

EXACT_OUTPUT

- EXACT_OUTPUT = 1

Defined in

[constants.ts:28](#)

id: overview sidebar_position: 1 title: Overview

Table of contents

Enumerations

- [Rounding](#)
- [SupportedChainId](#)
- [TradeType](#)

Classes

- [CurrencyAmount](#)
- [Ether](#)
- [Fraction](#)
- [NativeCurrency](#)
- [Percent](#)
- [Price](#)
- [Token](#)

Type Aliases

- [Bigintish](#)
- [Currency](#)

Variables

- [MaxUint256](#)
- [WETH9](#)

Functions

- [computePriceImpact](#)
- [sortedInsert](#)
- [sqrt](#)
- [validateAndParseAddress](#)

Type Aliases

Bigintish

T **Bigintish**: JSBI | string | number

Defined in

[constants.ts:24](#)

Currency

T **Currency**: [NativeCurrency](#) | [Token](#)

Defined in

[entities/currency.ts:4](#)

Variables

MaxUint256

- Const **MaxUint256**: default

Defined in

[constants.ts:37](#)

WETH9

- Const **WETH9**: Object

Known WETH9 implementation addresses, used in our implementation of Ether#wrapped

Index signature

- [chainId: number]: [Token](#)

Defined in

[entities/weth9.ts:6](#)

Functions

computePriceImpact

- **computePriceImpact**< TBase , TQuote >(midPrice , inputAmount , outputAmount): [Percent](#)

Returns the percent difference between the mid price and the execution price, i.e. price impact.

Type parameters

| Name | Type |
|--------|----------------------------------|
| TBase | extends Currency |
| TQuote | extends Currency |

Parameters

| Name | Type | Description |
|--------------|---|--------------------------------|
| midPrice | Price <TBase, TQuote> | mid price before the trade |
| inputAmount | CurrencyAmount <TBase> | the input amount of the trade |
| outputAmount | CurrencyAmount <TQuote> | the output amount of the trade |

Returns

[Percent](#)

Defined in

[utils/computePriceImpact.ts:9](#)

sortedInsert

▸ **sortedInsert**< T >(items , add , maxSize , comparator): T | null

Type parameters

| Name |
|------|
| T |

Parameters

| Name | Type |
|------------|------------------------|
| items | T[] |
| add | T |
| maxSize | number |
| comparator | (a: T, b: T) => number |

Returns

T | null

Defined in

[utils/sortedInsert.ts:5](#)

sqrt

▸ **sqrt**(value): JSBI

Computes floor(sqrt(value))

Parameters

| Name | Type | Description |
|-------|---------|--|
| value | default | the value for which to compute the square root, rounded down |

Returns

JSBI

Defined in

[utils/sqrt.ts:14](#)

validateAndParseAddress

▸ **validateAndParseAddress**(address): string

Validates an address and returns the parsed (checksummed) version of that address

Parameters

| Name | Type | Description |
|---------|--------|-------------------------------|
| address | string | the unchecksummed hex address |

Returns

string

Defined in

[utils/validateAndParseAddress.ts:7](#)

id: overview sidebar_position: 1 title: Overview

The Uniswap Swap Widget

Welcome to the Uniswap Swap Widget. To begin, we recommend looking at the [Guides](#) and for deeper reference see the [Swap Widget Github](#) repo.

Alpha software

The latest version of the Swap Widget is in production in the Uniswap Interface, but it is considered Alpha software and may contain bugs or change significantly between patch versions. If you have questions about how to use the SDK, please reach out in the `#dev-chat` channel of our Discord. Pull requests are welcome!

Uniswap Swap Widget

- [Swap Widget Github Repo](#)
- [Swap Widget NPM Package](#)



id: v1 title: API V1 Reference (Deprecated) sidebar_position: 2

Swap Widget API V1 Reference (Deprecated)

Required Parameters {#required-parameters}

| Prop Name | Prop Type | Default Value | Description |
|-----------------|-----------|---------------|---|
| jsonRpcEndpoint | string | undefined | URI of your JSON-RPC endpoint. Strongly recommended in order to provide trade quotes prior to the user connecting a wallet. If none is provided, the widget will be completely disabled until the user connects a wallet. Once a wallet is connected, the widget will use the wallet's JSON-RPC. See Understanding the Swap Widget States . |
| provider | any | undefined | An EIP-1193 provider. This is required to swap. |

Optional Parameters {#optional-parameters}

| Prop Name | Prop Type | Default Value | Description |
|---------------------------|-----------------------------|------------------------|---|
| convenienceFee | number | undefined | Optionally, you may charge a convenience fee on top of swaps executed through your web app. The allowed range is 1 to 100 basis points (inclusive of 100) consistent with the Uniswap v3 Periphery contract. |
| convenienceFeeRecipient | {[chainId: number]: string} | undefined | The address to receive the convenience fee on each network. Required if <code>convenienceFee</code> is provided. |
| defaultInputTokenAddress | {[chainId: number]: string} | string OR
'NATIVE' | Address of the token to be selected by default in the input field (e.g. USDC) for each network chain ID. If left empty the widget will use the native token of the connected chain as default. This can be explicitly defined by the special string ' <code>NATIVE</code> '. For convenience you may pass a single string instead of a <code>chainId</code> mapping. In this case, the widget will assume that string corresponds to an L1 Ethereum address with <code>chainId=1</code> . Any addresses provided in this parameter must be included in the <code>tokenList</code> . |
| defaultInputAmount | number | 0 | Default amount for the input field (e.g. 1 ETH). This value will respect the decimals of the <code>defaultInputTokenAddress</code> . This parameter is valid only if <code>defaultInputTokenAddress</code> is also set. This parameter is mutually exclusive with <code>defaultOutputAmount</code> , so you may set only one of <code>defaultInputAmount</code> and <code>defaultOutputAmount</code> . |
| defaultOutputTokenAddress | {[chainId: number]: string} | string OR
undefined | Address of the token to be selected by default in the input field (e.g. USDC) for each network chain ID. None if left empty. Any addresses provided in this parameter must be included in the <code>tokenList</code> . |

| | | | |
|---------------------|------------------|-------------|---|
| defaultOutputAmount | number | 0 | Default amount for the input field (e.g. 100 USDC). This value will respect the decimals of the <code>defaultOutputTokenAddress</code> . This parameter is mutually exclusive with <code>defaultInputAmount</code> , so you may set only one of <code>defaultInputAmount</code> and <code>defaultOutputAmount</code> . |
| locale | SupportedLocale | en-US | Specifies an explicit locale to use for the widget interface. This can be set to one of the values exported by the library in SUPPORTED LOCALES . |
| onConnectWallet | () => void | undefined | If passed, the “Connect your wallet” message will be clickable, and clicking it will trigger this handler function. This can be used to trigger your own wallet connection flow from the widget. |
| onError | ErrorHandler | undefined | An error handler which receives any errors that occur in the widget. This can be used for collecting error metrics. |
| theme | Theme | lightTheme | Specifies a custom theme (colors, font, and border radii). See Customizing the Theme . |
| tokenList | string | TokenInfo[] | Specifies the set of tokens that appear by default in the token selector list. Accepts either a URI of a token list as defined by the Token Lists standard, or an inline array of tokens. If none is provided, the Uniswap Labs default token list will be used. See Customizing the Default Token List . |
| width | number OR string | 360 | Specifies the width of the widget. If specified as a number, this is in pixels; otherwise, it is interpreted as a CSS <code><length></code> data type. Recommended width is 360px. Minimum width is 270px. See Customizing the Width . |

Subscribing to Events

During the lifecycle of the swap widget, most of the events you will need are available on the web3 provider. For example, the below snippet shows how to listen for events when the wallet account changes or a new wallet connects. You can see more event examples in the [MetaMask](#) docs.

```
// Subscribe to messages
interface ProviderMessage {
  type: string;
  data: unknown;
}

ethereum.on(
  'message',
  handler: (message: ProviderMessage) => void
);
```

:::note Questions? Join the [Discord channel](#) to ask questions and get support from the Uniswap community. :::

id: v2 title: API V2 Reference sidebar_position: 1

Swap Widget API V2 Reference

Recommended Parameters {#recommended-parameters}

| Prop Name | Prop Type | Default Value | Description |
|---------------|------------------------------------|-----------------------------|---|
| provider | JsonRpcProvider OR Eip1193Provider | undefined | An EIP-1193 provider. See Web3 provider . |
| jsonRpcUrlMap | { [chainId: number]: string[] } | JSON_RPC_FALLBACK_ENDPOINTS | Mapping of your JSON-RPC endpoint URLs indexed by chainId, used to provide trade quotes prior to the user connecting a wallet. If none are provided for a chain, the widget will fallback to public JSON-RPC endpoints, which are unreliable and rate-limited. See JSON-RPC Endpoints . |

Optional Parameters {#optional-parameters}

| Prop Name | Prop Type | Default Value | Description |
|-----------|-----------|---------------|-------------|
| | | | |

| | | | |
|---------------------------|---|------------------------|--|
| convenienceFee | number | undefined | Optionally, you may charge a convenience fee on top of swaps executed through your web app. The allowed range is 1 to 100 basis points paid in the output token of a swap, consistent with the Uniswap v3 Periphery contract. |
| convenienceFeeRecipient | {[chainId: number]: string} | undefined | The address to receive the convenience fee on each network. Required if <code>convenienceFee</code> is provided. |
| defaultChainId | number | 1 | You may specify which chainId you want to prompt a user to connect their wallet to. See a list of all chains supported on widget. |
| defaultInputTokenAddress | {[chainId: number]: string} | string OR
'NATIVE' | Address of the token to be selected by default in the input field (e.g. USDC) for each network chain ID. If left empty the widget will use the native token of the connected chain as default. This can be explicitly defined by the special string 'NATIVE'. For convenience you may pass a single string instead of a <code>chainId</code> mapping. In this case, the widget will assume that string corresponds to an L1 Ethereum address with <code>chainId=1</code> . Any addresses provided in this parameter must be included in the <code>tokenList</code> . |
| defaultInputAmount | number | 0 | Default amount for the input field (e.g. 1 ETH). This value will respect the decimals of the <code>defaultInputTokenAddress</code> . This parameter is valid only if <code>defaultInputTokenAddress</code> is also set. This parameter is mutually exclusive with <code>defaultOutputTokenAmount</code> , so you may set only one of <code>defaultInputTokenAmount</code> and <code>defaultOutputTokenAmount</code> . |
| defaultOutputTokenAddress | {[chainId: number]: string} | string OR
undefined | Address of the token to be selected by default in the input field (e.g. USDC) for each network chain ID. None if left empty. Any addresses provided in this parameter must be included in the <code>tokenList</code> . |
| defaultOutputAmount | number | 0 | Default amount for the input field (e.g. 100 USDC). This value will respect the decimals of the <code>defaultOutputTokenAddress</code> . This parameter is mutually exclusive with <code>defaultInputTokenAmount</code> , so you may set only one of <code>defaultInputTokenAmount</code> and <code>defaultOutputTokenAmount</code> . |
| hideConnectionUI | boolean | false | Hide the widget's built-in wallet connection UI, including the connected account chip & 'Connect wallet to swap' button. |
| locale | SupportedLocale | en-US | Specifies an explicit locale to use for the widget interface. This can be set to one of the values exported by the library in SUPPORTED LOCALES . |
| onConnectWalletClick | () => void OR () => Promise<boolean> | undefined | If passed, allows you to add custom behavior when the user clicks on the 'Connect your wallet to swap' button. To manage displaying the widget's built-in wallet connection modal, return a resolved promise with <code>resolve(true/false)</code> . |
| onSwitchChain | (addChainParameter:
AddEthereumChainParameter) =>
void OR Promise<void> | undefined | An integration hook called when the user tries to switch chains. If the hook returns a Promise, it is assumed the integrator is attempting to switch the chain, and no further attempts will be made. If that Promise rejects, the error will be ignored so as not to crash the widget. |
| onError | ErrorHandler | undefined | An error handler which receives any Javascript errors that occur in the widget. This can be used for collecting error metrics. |
| onReviewSwapClick | () => void OR () =>
Promise<boolean> | undefined | If passed, allows you to add custom behavior when the user clicks on the 'review swap' button. To manage progression to the review screen (i.e. to add a pre-swap warning speedbump), return a resolved promise with <code>resolve(true/false)</code> . |
| onTokenSelectorClick | (f: Field) => void (f:
Field) => Promise<boolean
void> | undefined | A click handler fired with the selected <code>Field</code> ('INPUT' 'OUTPUT') when the user clicks on a token selector dropdown. To manage progression to the native token |

| | | | |
|---------------|-------------------------------------|-------------|--|
| | | | selector view (i.e. to utilize your own external token selector UI), return a resolved promise with resolve(true/false). |
| onTxFail | (error: Error, data: any) => void | undefined | An error handler which receives error data for on-chain transaction failures. Does not include when user cancels a transaction or if a transaction isn't able to be submitted. |
| onTxSubmit | (txHash: string, data: any) => void | undefined | A handler that receives the transaction hash and related data when a user submits a transaction. |
| onTxSuccess | (txHash: string, data: any) => void | undefined | A handler that receives the transaction hash and related data when a transaction succeeds on-chain. |
| routerUrl | string | undefined | Optionally provide a base URL to your own hosted instance of the Uniswap Router API. If none is provided, the optimal trade route is computed by running the @uniswap/smart-order-router package locally in the browser; this can take a few seconds to load & is slower. You also may be able to find more optimal routes using the Router API! See more about deploying the Router API . |
| theme | Theme | lightTheme | Specifies a custom theme (colors, font, and border radii). See Customizing the Theme . |
| tokenList | string | TokenInfo[] | Specifies the set of tokens that appear by default in the token selector list. Accepts either a URI of a token list as defined by the Token Lists standard, or an inline array of tokens. If none is provided, the Uniswap Labs default token list will be used. See Customizing the Default Token List . |
| width | number OR string | 360 | Specifies the width of the widget. If specified as a number, this is in pixels; otherwise, it is interpreted as a CSS <code><length></code> data type. Recommended width is 360px. Minimum width is 300px. See Customizing the Width . |
| brandedFooter | boolean | true | Enables the "Powered by Uniswap" footer at the bottom of the widget. |
| dialog | HTMLDivElement | undefined | Specifies the element to portal widget dialogs (e.g. Summary, Transaction dialogs) into. |
| dialogOptions | DialogOptions | undefined | Specifies more custom dialog behavior, like transition animations. |

Subscribing to Events

During the lifecycle of the swap widget, most of the events you will need are available on the web3 provider. For example, the below snippet shows how to listen for events when the wallet account changes or a new wallet connects. You can see more event examples in the [MetaMask](#) docs.

```
// Subscribe to messages
interface ProviderMessage {
  type: string;
  data: unknown;
}

ethereum.on(
  'message',
  handler: (message: ProviderMessage) => void
);
```

:::note Questions? Join the [Discord channel](#) to ask questions and get support from the Uniswap community. :::

id: getting-started title: Getting Started sidebar_position: 1

The [Uniswap SDK](#) is meant to simplify every aspect of integrating Uniswap into your project. It's written in [TypeScript](#), has a [robust test suite](#), uses [bigumber.js](#) for math, and includes an optional data-fetching module which relies on [ethers.js](#).

The SDK was built to be extremely easy to use, but also feature-rich. It offers various levels of abstraction that make it suitable for use nearly anywhere, from hackathon projects to production applications.

Overview

The SDK is divided into several modular components that perform tightly scoped tasks:

- [Data](#) - Fetches Uniswap data from the blockchain
- [Computation](#) - Computes market- and trade-specific statistics using blockchain data
- [Format](#) - Formats data for display
- [Orchestration](#) - Offers named abstraction functions that seamlessly combine lower-level data- and computation-related functions
- [Transact](#) - Prepares computed trades for execution against Uniswap smart contracts
- [Constants](#) - Exports various helpful constants for use throughout the SDK

Additionally, it exports a number of custom types:

- [Types](#) - Exports all types used by the SDK

Installation

To start using the SDK, simply install it into your project...

```
yarn add @uniswap/sdk
```

...import some functions...

```
import { ... } from '@uniswap/sdk'
```

...and dive into the rest of the documentation to learn more!

id: overview sidebar_position: 1 title: Overview

The Uniswap V1 SDK

This is the original documentation for the Uniswap V1 SDK, released in 2019. The Uniswap V1 SDK is no longer under development, for the most recent implementation of the SDK, see the [V3 SDK](#).--- id: data title: Data

getTokenReserves

This function fetches Uniswap reserve data for a given token address on a given network.

- If only a chain id is specified, the Ethereum node used to fulfill data requests is determined by `ethers.getDefaultProvider`, else it is the one specified by the passed provider.
- This function throws an error if the provided tokenAddress is not a token with a Uniswap exchange.

Function Signature

```
export async function getTokenReserves(
  tokenAddress: string,
  chainIdOrProvider: ChainIdOrProvider = 1
): Promise<TokenReservesNormalized>
```

Input Parameters

| Parameter | Type | Description |
|-------------------|-------------------|---|
| tokenAddress | string | The checksummed address of a token with a Uniswap exchange. |
| chainIdOrProvider | ChainIdOrProvider | A supported chain id (1, 3, 4, or 42), or an underlying web3 provider connected to a chain with a supported chain id. |

Example Usage

```

const tokenAddress = '0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359' // DAI Mainnet
const chainIdOrProvider: ChainIdOrProvider = 1 // could be e.g. window.ethereum instead

const tokenReserves: TokenReservesNormalized = await getTokenReserves(tokenAddress, chainIdOrProvider)

/*
{
  // details for the passed token
  token: {
    chainId: 1,
    address: '0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359',
    decimals: 18
  },

  // details for the Uniswap exchange of the passed token
  exchange: {
    chainId: 1,
    address: '0x09cabEC1eAd1c0Ba254B09efb3EE13841712bE14',
    decimals: 18
  },

  // details for the ETH portion of the reserves of the passed token
  ethReserve: {
    token: {
      chainId: 1,
      address: 'ETH',
      decimals: 18
    },
    amount: <BigNumber>
  },

  // details for the token portion of the reserves of the passed token
  tokenReserve: {
    token: {
      chainId: 1,
      address: '0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359',
      decimals: 18
    },
    amount: <BigNumber>
  }
}
*/

```

id: computation title: Computation

getMarketDetails

This function computes market details for the passed reserves data. Markets are defined as ETH<>ERC20, ERC20<>ETH, or ERC20<>ERC20 pairs, where the first currency is the input and the second is the output. Reserves must be specified for both the input and output currency.

- In the case of ETH, `undefined` should be passed as the reserves data. `getTokenReserves` formatted ERC20 reserves, or the requisite data can be fetched manually and passed in.
- Rates are calculated to 18 decimal places of precision.

Function Signature

```

export function getMarketDetails(
  optionalReservesInput: OptionalReserves,
  optionalReservesOutput: OptionalReserves
): MarketDetails

```

Input Parameters

| Parameter | Type | Description |
|-----------|------|-------------|
|-----------|------|-------------|

| | | |
|------------------------|------------------|--|
| optionalReservesInput | OptionalReserves | Reserves data for the input currency. |
| optionalReservesOutput | OptionalReserves | Reserves data for the output currency. |

Example Usage

```

const reserves: ChainIdOrProvider = await getTokenReserves(tokenAddress)

const marketDetails: MarketDetails = getMarketDetails(undefined, reserves) // ETH<>ERC20

/*
{
  // market type
  tradeType: 'ETH_TO_TOKEN',

  // dummy ETH reserves
  inputReserves: {
    token: {
      chainId: 1,
      address: 'ETH',
      decimals: 18
    }
  },
  // normalized token reserves
  outputReserves: <NormalizedReserves>,

  // market rate calculated to 18 decimals of precision
  marketRate: {
    rate: <BigNumber>,           // x output / 1 input
    rateInverted: <BigNumber> // x input / 1 output
  }
}
*/

```

getTradeDetails

This function computes trade details for the passed market data.

-This function throws an error if the passed _tradeAmount is greater than the amount of ETH/tokens in the relevant Uniswap exchange.

- Trade amounts must be passed in non-decimal form (where e.g. 1 ETH is represented as 1000000000000000000000000 wei).

Function Signature

```

export function getTradeDetails(
  tradeExact: TRADE_EXACT,
  _tradeAmount: BigNumberish,
  marketDetails: MarketDetails
): TradeDetails

```

Input Parameters

| Parameter | Type | Description |
|---------------|---------------|--|
| tradeExact | TRADE_EXACT | Whether either the input or the output currency is the exact amount. |
| _tradeAmount | BigNumberish | The amount to buy/sell (of the output/input currency, depending on tradeExact) |
| marketDetails | MarketDetails | Market details. |

Example Usage

```

const _purchaseAmount: BigNumber = new BigNumber('2.5')
const _decimals: number = 18

```

```

const tradeAmount: BigNumber = _purchaseAmount.multipliedBy(10 ** _decimals)
const marketDetails: MarketDetails = getMarketDetails(undefined, reserves) // ETH<>ERC20

// buy exactly 2.5 of an 18 decimal ERC20 with ETH
const tradeDetails: TradeDetails = getTradeDetails(TRADE_EXACT.OUTPUT, tradeAmount, marketDetails)

/*
{
  marketDetailsPre: <MarketDetails>,
  marketDetailsPost: <MarketDetails>,
  tradeType: 'ETH_TO_TOKEN',
  tradeExact: 'OUTPUT',
  inputAmount: {
    token: <Token>,
    amount: <BigNumber>
  },
  outputAmount: {
    token: <Token>,
    amount: <BigNumber>
  },
  // execution rate calculated to 18 decimals of precision
  executionRate: {
    rate: <BigNumber>           // x output / 1 input
    rateInverted: <BigNumber> // x input / 1 output
  },
  // slippage between the pre- and post-trade market rates, in basis points, calculated to 18 decimals of precision
  marketRateSlippage: <BigNumber>,
  // slippage between the execution and pre-trade market rate, in basis points, calculated to 18 decimals of precision
  executionRateSlippage: <BigNumber>
}
*/

```

id: format title: Format

formatSignificant

This function formats values to a specified number of significant digits.

Function Signature

```
export function formatSignificant(bigNumberish: BigNumberish, options?: FormatSignificantOptions): string
```

Input Parameters

| Parameter | Type | Description |
|--------------|--------------------------|----------------------------|
| bigNumberish | BigNumberish | The value to be formatted. |
| options? | FormatSignificantOptions | Formatting options. |

Example Usage

```
const formatted: string = formatSignificant('123456', { significantDigits: 3 }) // 1.23
```

formatSignificantDecimals

This function formats token and ethereum values to a specified number of significant digits.

Function Signature

```
export function formatSignificantDecimals(  
  bigNumberish: BigNumberish,  
  decimals: number,  
  options?: FormatSignificantOptions  
): string
```

Input Parameters

| Parameter | Type | Description |
|--------------|--------------------------|-----------------------------------|
| bigNumberish | BigNumberish | The value to be formatted. |
| decimals | number | The decimals of the passed value. |
| options? | FormatSignificantOptions | Formatting options. |

Example Usage

```
const formatted: string = formatSignificantDecimals('12345600000000000000', 18, {  
  significantDigits: 3,  
}) // 1.23
```

formatFixed

This function formats values to a specified number of decimal places.

Function Signature

```
export function formatFixed(bigNumberish: BigNumberish, options?: FormatFixedOptions): string
```

Input Parameters

| Parameter | Type | Description |
|--------------|--------------------|----------------------------|
| bigNumberish | BigNumberish | The value to be formatted. |
| options? | FormatFixedOptions | Formatting options. |

Example Usage

```
const formatted: string = formatFixed('1.2345', { decimalPlaces: 2 }) // 1.23
```

formatFixedDecimals

This function formats token and ethereum values to a specified number of decimal places.

Function Signature

```
export function formatFixedDecimals(bigNumberish: BigNumberish, decimals: number, options?: FormatFixedOptions):  
string
```

Input Parameters

| Parameter | Type | Description |
|--------------|--------------------|-----------------------------------|
| bigNumberish | BigNumberish | The value to be formatted. |
| decimals | number | The decimals of the passed value. |
| options? | FormatFixedOptions | Formatting options. |

Example Usage

```
const formatted: string = formatFixedDecimals('1234560000000000000000', 18, {
  decimalPlaces: 2,
}) // 1.23
```

id: orchestration title: Orchestration

Orchestration functions are plain-english wrappers for the function defined in [/sdk/1.0.0/reference/data](#) and [Computation](#).

Functions suffixed with `WithData` are synchronous, and require token reserves to be passed in as arguments. Functions without the suffix are asynchronous, and require token addresses to be passed in as arguments.

tradeExactEthForTokensWithData

The function facilitates trading an exact amount of ETH for a specified token.

Function Signature

```
export function tradeExactEthForTokensWithData(reserves: OptionalReserves, ethAmount: BigNumberish): TradeDetails
```

Input Parameters

| Parameter | Type | Description |
|-----------|------------------|-------------------------------------|
| reserves | OptionalReserves | Reserves data for the output token. |
| ethAmount | BigNumberish | The input amount of ETH. |

Example Usage

```
const tradeDetails: TradeDetails = tradeExactEthForTokensWithData(reserves, '10000000000000000000000000000000')
```

tradeExactEthForTokens

The function facilitates trading an exact amount of ETH for a specified token.

Function Signature

```
export async function tradeExactEthForTokens(
  tokenAddress: string,
  ethAmount: BigNumberish,
  chainIdOrProvider?: ChainIdOrProvider
): Promise<TradeDetails>
```

Input Parameters

| Parameter | Type | Description |
|--------------|--------------|--------------------------|
| tokenAddress | string | Address of output token. |
| ethAmount | BigNumberish | The input amount of ETH. |

| | | |
|--------------------|-------------------|---|
| chainIdOrProvider? | ChainIdOrProvider | A supported chain id (1, 3, 4, or 42), or an underlying web3 provider connected to a chain with a supported chain id. |
|--------------------|-------------------|---|

Example Usage

```
const tradeDetails: TradeDetails = await tradeExactEthForTokens(tokenAddress, '10000000000000000000')
```

tradeEthForExactTokensWithData

The function facilitates trading ETH for an exact amount of a specified token.

Function Signature

```
export function tradeEthForExactTokensWithData(reserves: OptionalReserves, tokenAmount: BigNumberish): TradeDetails
```

Input Parameters

| Parameter | Type | Description |
|-------------|------------------|-------------------------------------|
| reserves | OptionalReserves | Reserves data for the output token. |
| tokenAmount | BigNumberish | The output amount of tokens. |

Example Usage

```
const tradeDetails: TradeDetails = tradeEthForExactTokensWithData(reserves, '10000000000000000000')
```

tradeEthForExactTokens

The function facilitates trading ETH for an exact amount of a specified token.

Function Signature

```
export async function tradeEthForExactTokens(
  tokenAddress: string,
  tokenAmount: BigNumberish,
  chainIdOrProvider?: ChainIdOrProvider
): Promise<TradeDetails>
```

Input Parameters

| Parameter | Type | Description |
|--------------------|-------------------|---|
| tokenAddress | string | Address of output token. |
| tokenAmount | BigNumberish | The output amount of tokens. |
| chainIdOrProvider? | ChainIdOrProvider | A supported chain id (1, 3, 4, or 42), or an underlying web3 provider connected to a chain with a supported chain id. |

Example Usage

```
const tradeDetails: TradeDetails = await tradeEthForExactTokens(tokenAddress, '10000000000000000000')
```

tradeExactTokensForEthWithData

The function facilitates trading an exact amount of a specified token for ETH.

Function Signature

```
export function tradeExactTokensForEthWithData(reserves: OptionalReserves, tokenAmount: BigNumberish): TradeDetails
```

Input Parameters

| Parameter | Type | Description |
|-------------|------------------|------------------------------------|
| reserves | OptionalReserves | Reserves data for the input token. |
| tokenAmount | BigNumberish | The input amount of tokens. |

Example Usage

```
const tradeDetails: TradeDetails = tradeExactTokensForEthWithData(reserves, '10000000000000000000')
```

tradeExactTokensForEth

The function facilitates trading an exact amount of a specified token for ETH.

Function Signature

```
export async function tradeExactTokensForEth(
  tokenAddress: string,
  tokenAmount: BigNumberish,
  chainIdOrProvider?: ChainIdOrProvider
): Promise<TradeDetails>
```

Input Parameters

| Parameter | Type | Description |
|--------------------|-------------------|---|
| tokenAddress | string | Address of input token. |
| tokenAmount | BigNumberish | The input amount of tokens. |
| chainIdOrProvider? | ChainIdOrProvider | A supported chain id (1, 3, 4, or 42), or an underlying web3 provider connected to a chain with a supported chain id. |

Example Usage

```
const tradeDetails: TradeDetails = await tradeExactTokensForEth(tokenAddress, '10000000000000000000')
```

tradeTokensForExactEthWithData

The function facilitates trading a specified token for an exact amount of ETH.

Function Signature

```
export function tradeTokensForExactEthWithData(reserves: OptionalReserves, ethAmount: BigNumberish): TradeDetails
```

Input Parameters

| Parameter | Type | Description |
|-----------|------------------|------------------------------------|
| reserves | OptionalReserves | Reserves data for the input token. |
| ethAmount | BigNumberish | The output amount of ETH. |

Example Usage

```
const tradeDetails: TradeDetails = tradeTokensForExactEthWithData(reserves, '10000000000000000000')
```

tradeTokensForExactEth

The function facilitates trading a specified token for an exact amount of ETH.

Function Signature

```
export async function tradeTokensForExactEth(
  tokenAddress: string,
  ethAmount: BigNumberish,
  chainIdOrProvider?: ChainIdOrProvider
): Promise<TradeDetails>
```

Input Parameters

| Parameter | Type | Description |
|--------------------|-------------------|---|
| tokenAddress | string | Address of input token. |
| ethAmount | BigNumberish | The output amount of ETH. |
| chainIdOrProvider? | ChainIdOrProvider | A supported chain id (1, 3, 4, or 42), or an underlying web3 provider connected to a chain with a supported chain id. |

Example Usage

```
const tradeDetails: TradeDetails = await tradeTokensForExactEth(tokenAddress, '10000000000000000000')
```

tradeExactTokensForTokensWithData

The function facilitates trading an exact amount of a specified token for another token.

Function Signature

```
export function tradeExactTokensForTokensWithData(
  reservesInput: OptionalReserves,
  reservesOutput: OptionalReserves,
  tokenAmount: BigNumberish
): TradeDetails
```

Input Parameters

| Parameter | Type | Description |
|----------------|------------------|-------------------------------------|
| reservesInput | OptionalReserves | Reserves data for the input token. |
| reservesOutput | OptionalReserves | Reserves data for the output token. |
| tokenAmount | BigNumberish | The input amount of tokens. |

Example Usage

```
const tradeDetails: TradeDetails = tradeExactTokensForTokensWithData(
  reservesInput,
  reservesOutput,
  '10000000000000000000'
)
```

tradeExactTokensForTokens

The function facilitates trading an exact amount of a specified token for another token.

Function Signature

```
export async function tradeExactTokensForTokens(
  tokenAddressInput: string,
  tokenAddressOutput: string,
  tokenAmount: BigNumberish,
  chainIdOrProvider?: ChainIdOrProvider
): Promise<TradeDetails>
```

Input Parameters

| Parameter | Type | Description |
|--------------------|-------------------|---|
| tokenAddressInput | string | Address of input token. |
| tokenAddressOutput | string | Address of output token. |
| tokenAmount | BigNumberish | The input amount of tokens. |
| chainIdOrProvider? | ChainIdOrProvider | A supported chain id (1, 3, 4, or 42), or an underlying web3 provider connected to a chain with a supported chain id. |

Example Usage

```
const tradeDetails: TradeDetails = await tradeExactTokensForTokens(
  tokenAddressInput,
  tokenAddressOutput,
  '1000000000000000000'
)
```

tradeTokensForExactTokensWithData

The function facilitates trading a specified token for an exact amount of another token.

Function Signature

```
export function tradeTokensForExactTokensWithData(
  reservesInput: OptionalReserves,
  reservesOutput: OptionalReserves,
  tokenAmount: BigNumberish
): TradeDetails
```

Input Parameters

| Parameter | Type | Description |
|----------------|------------------|-------------------------------------|
| reservesInput | OptionalReserves | Reserves data for the input token. |
| reservesOutput | OptionalReserves | Reserves data for the output token. |
| tokenAmount | BigNumberish | The output amount of tokens. |

Example Usage

```
const tradeDetails: TradeDetails = tradeTokensForExactTokensWithData(
  reservesInput,
  reservesOutput,
```

```
'10000000000000000000000000000000'  
)
```

tradeTokensForExactTokens

The function facilitates trading an exact amount of a specified token for another token.

Function Signature

```
export async function tradeTokensForExactTokens(  
    tokenAddressInput: string,  
    tokenAddressOutput: string,  
    tokenAmount: BigNumberish,  
    chainIdOrProvider?: ChainIdOrProvider  
): Promise<TradeDetails>
```

Input Parameters

| Parameter | Type | Description |
|--------------------|-------------------|---|
| tokenAddressInput | string | Address of input token. |
| tokenAddressOutput | string | Address of output token. |
| tokenAmount | BigNumberish | The output amount of tokens. |
| chainIdOrProvider? | ChainIdOrProvider | A supported chain id (1, 3, 4, or 42), or an underlying web3 provider connected to a chain with a supported chain id. |

Example Usage

```
const tradeDetails: TradeDetails = await tradeTokensForExactTokens(  
    tokenAddressInput,  
    tokenAddressOutput,  
    '10000000000000000000000000000000'  
)
```

id: transact title: Transact

getExecutionDetails

The function formats trade data for execution against the relevant Uniswap exchange.

Function Signature

```
export function getExecutionDetails(  
    trade: TradeDetails,  
    maxSlippage?: number,  
    deadline?: number,  
    recipient?: string  
): ExecutionDetails
```

Input Parameters

| Parameter | Type | Description |
|--------------|--------------|---|
| trade | TradeDetails | The trade to execute. |
| maxSlippage? | number | The maximum slippage to allow, in basis points. Defaults to 200 (2%). |
| deadline? | number | When the transaction will expire. Defaults to 10 minutes in the future. |
| recipient? | number | An optional recipient address. Defaults to the msg.sender |

Example Usage

Method arguments are returned as one of: `BigNumber`, `number`, or `string`. `BigNumber`s are large number objects, `numbers` are small numbers in base 10, and `string`s are addresses.

```
const tradeDetails: TradeDetails = tradeExactEthForTokensWithData(reserves, '10000000000000000000')

const executionDetails: ExecutionDetails = await getExecutionDetails(tradeDetails)

/*
{
  // the address of the relevant exchange
  exchangeAddress: '0x09cabEC1eAd1c0Ba254B09efb3EE13841712bE14',

  // the name of the method that must be called
  methodName: "ethToTokenSwapInput",

  // the id of the method name
  methodId: "0xf39b5b9b",

  // the ether value that must be sent with the transaction
  value: <BigNumber>,

  // method arguments as an array
  methodArguments: MethodArgument[]
}
*/
```

id: constants title: Constants

Below is an exhaustive list of all external constants used in the SDK.

```
import BigNumber from 'bignumber.js'

import ERC20 from './abis/ERC20.json'
import FACTORY from './abis/FACTORY.json'
import EXCHANGE from './abis/EXCHANGE.json'

export const ETH = 'ETH'

export enum SUPPORTED_CHAIN_ID {
  Mainnet = 1,
  Ropsten = 3,
  Rinkeby = 4,
  Kovan = 42,
}

export const FACTORY_ADDRESS: { [key: number]: string } = {}

export const FACTORY_ABI: string = JSON.stringify(FACTORY)
export const EXCHANGE_ABI: string = JSON.stringify(EXCHANGE)

export enum TRADE_TYPE {
  ETH_TO_TOKEN = 'ETH_TO_TOKEN',
  TOKEN_TO_ETH = 'TOKEN_TO_ETH',
  TOKEN_TO_TOKEN = 'TOKEN_TO_TOKEN',
}

export enum TRADE_EXACT {
  INPUT = 'INPUT',
  OUTPUT = 'OUTPUT',
}

export enum TRADE_METHODS {
  ethToTokenSwapInput = 'ethToTokenSwapInput',
  ethToTokenTransferInput = 'ethToTokenTransferInput',
  ethToTokenSwapOutput = 'ethToTokenSwapOutput',
  ethToTokenTransferOutput = 'ethToTokenTransferOutput',
}
```

```

    tokenToEthSwapInput = 'tokenToEthSwapInput',
    tokenToEthTransferInput = 'tokenToEthTransferInput',
    tokenToEthSwapOutput = 'tokenToEthSwapOutput',
    tokenToEthTransferOutput = 'tokenToEthTransferOutput',
    tokenToTokenSwapInput = 'tokenToTokenSwapInput',
    tokenToTokenTransferInput = 'tokenToTokenTransferInput',
    tokenToTokenSwapOutput = 'tokenToTokenSwapOutput',
    tokenToTokenTransferOutput = 'tokenToTokenTransferOutput',
}

export const TRADE_METHOD_IDS: { [key: string]: string } = {}

export enum FIXED_UNDERFLOW_BEHAVIOR {
    ZERO = 'ZERO',
    LESS_THAN = 'LESS_THAN',
    ONE_DIGIT = 'ONE_DIGIT',
}

```

id: types title: Types

Below is an exhaustive list of all the external types used in the SDK.

```

import BigNumber from 'bignumber.js'
import { ethers } from 'ethers'

import { SUPPORTED_CHAIN_ID, TRADE_TYPE, TRADE_EXACT, FIXED_UNDERFLOW_BEHAVIOR } from './constants'

export type BigNumberish = BigNumber | ethers.utils.BigNumber | string | number

/// types for on-chain, submitted, and normalized data
export type ChainIdOrProvider = SUPPORTED_CHAIN_ID | ethers.providers.AsyncSendable | ethers.providers.Provider

// type guard for ChainIdOrProvider
export function isChainId(chainIdOrProvider: ChainIdOrProvider): chainIdOrProvider is SUPPORTED_CHAIN_ID {
    const chainId: SUPPORTED_CHAIN_ID = chainIdOrProvider as SUPPORTED_CHAIN_ID
    return typeof chainId === 'number'
}

// type guard for ChainIdOrProvider
export function isLowLevelProvider(
    chainIdOrProvider: ChainIdOrProvider
): chainIdOrProvider is ethers.providers.AsyncSendable {
    if (isChainId(chainIdOrProvider)) {
        return false
    } else {
        const provider: ethers.providers.AsyncSendable = chainIdOrProvider as ethers.providers.AsyncSendable
        return 'send' in provider || 'sendAsync' in provider
    }
}

export interface Token {
    chainId?: SUPPORTED_CHAIN_ID
    address?: string
    decimals: number
}

export interface TokenAmount {
    token: Token
    amount: BigNumberish
}

export interface TokenAmountNormalized {
    token: Token
    amount: BigNumber
}

export interface TokenReserves {
    token: Token
    exchange?: Token
}

```

```

    ethReserve: TokenAmount
    tokenReserve: TokenAmount
}

export interface TokenReservesNormalized {
    token: Token
    exchange?: Token
    ethReserve: TokenAmountNormalized
    tokenReserve: TokenAmountNormalized
}

export interface EthReserves {
    token: Token
}

// type for input data
export type OptionalReserves = TokenReserves | EthReserves | undefined

// type guard for OptionalReserves
export function areTokenReserves(reserves: OptionalReserves): reserves is TokenReserves {
    const tokenReserves: TokenReserves = reserves as TokenReserves
    return (
        tokenReserves !== undefined && tokenReserves.ethReserve !== undefined && tokenReserves.tokenReserve !==
undefined
    )
}

// type guard for OptionalReserves
export function areETHReserves(reserves: OptionalReserves): reserves is EthReserves {
    const tokenReserves: TokenReserves = reserves as TokenReserves
    return (
        tokenReserves !== undefined && tokenReserves.ethReserve === undefined && tokenReserves.tokenReserve ===
undefined
    )
}

// type for output data
export type NormalizedReserves = TokenReservesNormalized | EthReserves

// type guard for NormalizedReserves
export function areTokenReservesNormalized(reserves: NormalizedReserves): reserves is TokenReservesNormalized {
    const tokenReservesNormalized: TokenReservesNormalized = reserves as TokenReservesNormalized
    return tokenReservesNormalized.ethReserve !== undefined && tokenReservesNormalized.tokenReserve !== undefined
}

//// types for computed data
export interface Rate {
    rate: BigNumber
    rateInverted: BigNumber
}
export interface MarketDetails {
    tradeType: TRADE_TYPE
    inputReserves: NormalizedReserves
    outputReserves: NormalizedReserves
    marketRate: Rate
}

export interface TradeDetails {
    marketDetailsPre: MarketDetails
    marketDetailsPost: MarketDetails
    tradeType: TRADE_TYPE
    tradeExact: TRADE_EXACT
    inputAmount: TokenAmountNormalized
    outputAmount: TokenAmountNormalized
    executionRate: Rate
    marketRateSlippage: BigNumber
    executionRateSlippage: BigNumber
}

export type MethodArgument = BigNumber | number | string

```

```

export interface ExecutionDetails {
  exchangeAddress: string
  methodName: string
  methodId: string
  value: BigNumber
  methodArguments: MethodArgument[]
}

//// types for formatting data
export type FlexibleFormat = BigNumber.Format | boolean

// type guard for FlexibleFormat
export function isFormat(flexibleFormat: FlexibleFormat): flexibleFormat is BigNumber.Format {
  const format: BigNumber.Format = flexibleFormat as BigNumber.Format
  return typeof format !== 'boolean'
}

export interface FormatSignificantOptions {
  significantDigits: number
  roundingMode: BigNumber.RoundingMode
  forceIntegerSignificance: boolean
  format: FlexibleFormat
}

export interface FormatFixedOptions {
  decimalPlaces: number
  roundingMode: BigNumber.RoundingMode
  dropTrailingZeros: boolean
  underflowBehavior: FIXED_UNDERFLOW_BEHAVIOR
  format: FlexibleFormat
}

```

id: quick-start title: SDK Quick start

The Uniswap SDK exists to help developers build on top of Uniswap. It's designed to run in any environment that can execute JavaScript (think websites, node scripts, etc.). While simple enough to use in a hackathon project, it's also robust enough to power production applications.

Installation

The easiest way to consume the SDK is via [npm](#). To install it in your project, simply run `yarn add @uniswap/sdk` (or `npm install @uniswap/sdk`).

Usage

To run code from the SDK in your application, use an `import` or `require` statement, depending on which your environment supports. Note that the guides following this page will use ES6 syntax.

ES6 (`import`)

```

import { ChainId } from '@uniswap/sdk'
console.log(`The chainId of mainnet is ${ChainId.MAINNET}.`)

```

CommonJS (`require`)

```

const UNISWAP = require('@uniswap/sdk')
console.log(`The chainId of mainnet is ${UNISWAP.ChainId.MAINNET}.`)

```

Reference

Comprehensive reference material for the SDK is publicly available on the [Uniswap Labs Github](#).

id: fetching-data title: Fetching Data

Looking for a [quickstart](#)?

While the SDK is fully self-contained, there are two cases where it needs *on-chain data* to function. This guide will detail both of these cases, and offer some strategies that you can use to fetch this data.

Case 1: Tokens

Unsurprisingly, the SDK needs some notion of an ERC-20 token to be able to function. This immediately raises the question of *where data about tokens comes from*.

As an example, let's try to represent DAI in a format the SDK can work with. To do so, we need at least 3 pieces of data: a **chainId**, a **token address**, and how many **decimals** the token has. We also may be interested in the **symbol** and/or **name** of the token.

Identifying Data

The first two pieces of data — **chainId** and **token address** — must be provided by us. Thinking about it, this makes sense, as there's really no other way to unambiguously identify a token.

So, in the case of DAI, we know that the **chainId** is `1` (we're on mainnet), and the **token address** is

`0x6B175474E89094C44Da98b954EedeAC495271d0F`. Note that it's very important to externally verify token addresses. Don't use addresses from sources you don't trust!

Required Data

The next piece of data we need is **decimals**.

Provided by the User

One option here is to simply pass in the correct value, which we may know is `18`. At this point, we're ready to represent DAI as a [Token](#):

```
import { ChainId, Token } from '@uniswap/sdk'

const chainId = ChainId.MAINNET
const tokenAddress = '0x6B175474E89094C44Da98b954EedeAC495271d0F' // must be checksummed
const decimals = 18

const DAI = new Token(chainId, tokenAddress, decimals)
```

If we don't know or don't want to hardcode the value, we could look it up ourselves via any method of retrieving on-chain data in a function that looks something like:

```
import { ChainId } from '@uniswap/sdk'

async function getDecimals(chainId: ChainId, tokenAddress: string): Promise<number> {
  // implementation details
}
```

Fetched by the SDK

If we don't want to provide or look up the value ourselves, we can ask the SDK to look it up for us with [Fetcher.fetchTokenData](#)

```
import { ChainId, Token, Fetcher } from '@uniswap/sdk'

const chainId = ChainId.MAINNET
const tokenAddress = '0x6B175474E89094C44Da98b954EedeAC495271d0F' // must be checksummed

// note that you may want/need to handle this async code differently,
// for example if top-level await is not an option
const DAI: Token = await Fetcher.fetchTokenData(chainId, tokenAddress)
```

By default, this method will use the [default provider defined by ethers.js](#). If you're already using ethers.js in your application, you may pass in your provider as a 3rd argument. If you're using another library, you'll have to fetch the data separately.

Optional Data

Finally, we can talk about **symbol** and **name**. Because these fields aren't used anywhere in the SDK itself, they're optional, and can be provided if you want to use them in your application. However, the SDK will not fetch them for you, so you'll have to provide them:

```
import { ChainId, Token } from '@uniswap/sdk'

const DAI = new Token(ChainId.MAINNET, '0x6B175474E89094C44Da98b954EedeAC495271d0F', 18, 'DAI', 'Dai Stablecoin')
```

or:

```
import { ChainId, Token, Fetcher } from '@uniswap/sdk'

// note that you may want/need to handle this async code differently,
// for example if top-level await is not an option
const DAI = await Fetcher.fetchTokenData(
  ChainId.MAINNET,
  '0x6B175474E89094C44Da98b954EedeAC495271d0F',
  undefined,
  'DAI',
  'Dai Stablecoin'
)
```

Case 2: Pairs

Now that we've explored how to define a token, let's talk about pairs. To read more about what Uniswap pairs are, see [Pair](#).

As an example, let's try to represent the DAI-WETH pair.

Identifying Data

Each pair consists of two tokens (see previous section). Note that WETH used by the router is [exported by the SDK](#).

Required Data

The data we need is the *reserves* of the pair. To read more about reserves, see [getReserves](#).

Provided by the User

One option here is to simply pass in values which we've fetched ourselves to create a [Pair](#):

```
import { ChainId, Token, WETH, Pair, TokenAmount } from '@uniswap/sdk'

const DAI = new Token(ChainId.MAINNET, '0x6B175474E89094C44Da98b954EedeAC495271d0F', 18)

async function getPair(): Promise<Pair> {
  const pairAddress = Pair.getAddress(DAI, WETH[DAI.chainId])

  const reserves = [
    /* use pairAddress to fetch reserves here */
  ]
  const [reserve0, reserve1] = reserves

  const tokens = [DAI, WETH[DAI.chainId]]
  const [token0, token1] = tokens[0].sortsBefore(tokens[1]) ? tokens : [tokens[1], tokens[0]]

  const pair = new Pair(new TokenAmount(token0, reserve0), new TokenAmount(token1, reserve1))
  return pair
}
```

Note that these values can change as frequently as every block, and should be kept up-to-date.

Fetched by the SDK

If we don't want to look up the value ourselves, we can ask the SDK to look them up for us with [Fetcher.fetchTokenData](#):

```
import { ChainId, Token, WETH, Fetcher } from '@uniswap/sdk'

const DAI = new Token(ChainId.MAINNET, '0x6B175474E89094C44Da98b954EedeAC495271d0F', 18)
```

```
// note that you may want/need to handle this async code differently,
// for example if top-level await is not an option
const pair = await Fetcher.fetchPairData(DAI, WETH[DAI.chainId])
```

By default, this method will use the [default provider defined by ethers.js](#). If you're already using ethers.js in your application, you may pass in your provider as a 3rd argument. If you're using another library, you'll have to fetch the data separately.

Note that these values can change as frequently as every block, and should be kept up-to-date.

id: pricing title: Pricing

Looking for a [quickstart](#)?

Let's talk pricing. This guide will focus on the two most important Uniswap prices: the **mid price** and the **execution price**.

Mid Price

The mid price, in the context of Uniswap, is the price that reflects the *ratio of reserves in one or more pairs*. There are three ways we can think about this price. Perhaps most simply, it defines the relative value of one token in terms of the other. It also represents the price at which you could theoretically trade an infinitesimal amount (ϵ) of one token for the other. Finally, it can be interpreted as the current *market-clearing or fair value price* of the assets.

Let's consider the mid price for DAI-WETH (that is, the amount of DAI per 1 WETH).

Direct

The simplest way to get the DAI-WETH mid price is to observe the pair directly:

```
import { ChainId, Token, WETH, Fetcher, Route } from '@uniswap/sdk'

const DAI = new Token(ChainId.MAINNET, '0x6B175474E89094C44Da98b954EedeAC495271d0F', 18)

// note that you may want/need to handle this async code differently,
// for example if top-level await is not an option
const pair = await Fetcher.fetchPairData(DAI, WETH[DAI.chainId])

const route = new Route([pair], WETH[DAI.chainId])

console.log(route.midPrice.toSignificant(6)) // 201.306
console.log(route.midPrice.invert().toSignificant(6)) // 0.00496756
```

You may be wondering why we have to construct a *route* to get the mid price, as opposed to simply getting it from the pair (which, after all, includes all the necessary data). The reason is simple: a route forces us to be opinionated about the *direction* of trading. Routes consist of one or more pairs, and an input token (which fully defines a trading path). In this case, we passed WETH as the input token, meaning we're interested in a WETH \rightarrow DAI trade.

Now we understand that the mid price is going to be defined in terms of DAI/WETH. Not to worry though, if we need the WETH/DAI price, we can easily invert.

Finally, you may have noticed that we're formatting the price to 6 significant digits. This is because internally, prices are stored as exact-precision fractions, which can be converted to other representations on demand. For a full list of options, see [Price](#).

Indirect

For the sake of example, let's imagine a direct pair between DAI and WETH *doesn't exist*. In order to get a DAI-WETH mid price we'll need to pick a valid route. Imagine both DAI and WETH have pairs with a third token, USDC. In that case, we can calculate an indirect mid price through the USDC pairs:

```
import { ChainId, Token, WETH, Fetcher, Route } from '@uniswap/sdk'

const USDC = new Token(ChainId.MAINNET, '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48', 6)
const DAI = new Token(ChainId.MAINNET, '0x6B175474E89094C44Da98b954EedeAC495271d0F', 18)

// note that you may want/need to handle this async code differently,
// for example if top-level await is not an option
const USDCWETHPair = await Fetcher.fetchPairData(USDC, WETH[ChainId.MAINNET])
const DAIUSDCPair = await Fetcher.fetchPairData(DAI, USDC)

const route = new Route([USDCWETHPair, DAIUSDCPair], WETH[ChainId.MAINNET])
```

```
console.log(route.midPrice.toSignificant(6)) // 202.081
console.log(route.midPrice.invert().toSignificant(6)) // 0.00494851
```

Execution Price

Mid prices are great representations of the *current* state of a route, but what about trades? It turns out that it makes sense to define another price, the *execution* price of a trade, as the ratio of assets sent/received.

Imagine we're interested in trading 1 WETH for DAI:

```
import { ChainId, Token, WETH, Fetcher, Trade, Route, TokenAmount, TradeType } from '@uniswap/sdk'

const DAI = new Token(ChainId.MAINNET, '0x6B175474E89094C44Da98b954EedeAC495271d0F', 18)

// note that you may want/need to handle this async code differently,
// for example if top-level await is not an option
const pair = await Fetcher.fetchPairData(DAI, WETH[DAI.chainId])

const route = new Route([pair], WETH[DAI.chainId])

const trade = new Trade(route, new TokenAmount(WETH[DAI.chainId], '1000000000000000000'), TradeType.EXACT_INPUT)

console.log(trade.executionPrice.toSignificant(6))
console.log(trade.nextMidPrice.toSignificant(6))
```

Notice that we're constructing a trade of 1 WETH for as much DAI as possible, *given the current reserves of the direct pair*. The execution price represents the average DAI/WETH price for this trade. Of course, the reserves of any pair can change every block, which would affect the execution price.

Also notice that we're able to access the *next mid price*, if the trade were to complete successfully before the reserves changed.

id: trading title: Trading

Looking for a [quickstart](#)?

The SDK *cannot execute trades or send transactions on your behalf*. Rather, it offers utility classes and functions which make it easy to calculate the data required to safely interact with Uniswap. Nearly everything you need to safely transact with Uniswap is provided by the [Trade](#) entity. However, it is your responsibility to use this data to send transactions in whatever context makes sense for your application.

This guide will focus exclusively on sending a transaction to the [currently recommended Uniswap router](#)

Sending a Transaction to the Router

Let's say we want to trade 1 WETH for as much DAI as possible:

```
import { ChainId, Token, WETH, Fetcher, Trade, Route, TokenAmount, TradeType } from '@uniswap/sdk'

const DAI = new Token(ChainId.MAINNET, '0x6B175474E89094C44Da98b954EedeAC495271d0F', 18)

// note that you may want/need to handle this async code differently,
// for example if top-level await is not an option
const pair = await Fetcher.fetchPairData(DAI, WETH[DAI.chainId])

const route = new Route([pair], WETH[DAI.chainId])

const amountIn = '1000000000000000000' // 1 WETH

const trade = new Trade(route, new TokenAmount(WETH[DAI.chainId], amountIn), TradeType.EXACT_INPUT)
```

So, we've constructed a trade entity, but how do we use it to actually send a transaction? There are still a few pieces we need to put in place.

Before going on, we should explore how ETH works in the context of trading. Internally, the SDK uses WETH, as all Uniswap V2 pairs use WETH under the hood. However, it's perfectly possible for you as an end user to use ETH, and rely on the router to handle converting to/from WETH. So, let's use ETH.

The first step is selecting the appropriate router function. The names of router functions are intended to be self-explanatory; in this case we want [swapExactETHForTokens](#), because we're swapping an exact amount of ETH for tokens.

That Solidity interface for this function is:

```
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
    external
    payable
    returns (uint[] memory amounts);
```

Jumping back to our trading code, we can construct all the necessary parameters:

```
import { Percent } from '@uniswap/sdk'

const slippageTolerance = new Percent('50', '10000') // 50 bips, or 0.50%

const amountOutMin = trade.minimumAmountOut(slippageTolerance).raw // needs to be converted to e.g. hex
const path = [WETH[DAI.chainId].address, DAI.address]
const to = '' // should be a checksummed recipient address
const deadline = Math.floor(Date.now() / 1000) + 60 * 20 // 20 minutes from the current Unix time
const value = trade.inputAmount.raw // // needs to be converted to e.g. hex
```

The slippage tolerance encodes *how large of a price movement we're willing to tolerate before our trade will fail to execute*. Since Ethereum transactions are broadcast and confirmed in an adversarial environment, this tolerance is the best we can do to protect ourselves against price movements. We use this slippage tolerance to calculate the *minimum* amount of DAI we must receive before our trade reverts, thanks to [minimumAmountOut](#). Note that this code calculates this worst-case outcome assuming that the current price, i.e the route's *mid* price, is fair (usually a good assumption because of arbitrage).

The path is simply the ordered list of token addresses we're trading through, in our case WETH and DAI (note that we use the WETH address, even though we're using ETH).

The to address is the address that will receive the DAI.

The deadline is the Unix timestamp after which the transaction will fail, to protect us in the case that our transaction takes a long time to confirm and we wish to rescind our trade.

The value is the amount of ETH that must be included as the `msg.value` in our transaction.

id: getting-pair-addresses title: Pair Addresses

getPair

The most obvious way to get the address for a pair is to call [getPair](#) on the factory. If the pair exists, this function will return its address, else `address(0) (0x00)`.

- The "canonical" way to determine whether or not a pair exists.
- Requires an on-chain lookup.

CREATE2

Thanks to some [fancy footwork in the factory](#), we can also compute pair addresses without any on-chain lookups because of [CREATE2](#). The following values are required for this technique:

| | |
|----------------------|---|
| address | The factory address |
| salt | <code>keccak256(abi.encodePacked(token0, token1))</code> |
| keccak256(init_code) | <code>0x96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f</code> |

- `token0` must be strictly less than `token1` by sort order.
- Can be computed offline.
- Requires the ability to perform `keccak256`.

Examples

TypeScript

This example makes use of the [Uniswap SDK](#). In reality, the SDK computes pair addresses behind the scenes, obviating the need to compute them manually like this.

```
import { FACTORY_ADDRESS, INIT_CODE_HASH } from '@uniswap/sdk'
import { pack, keccak256 } from '@ethersproject/solidity'
import { getCreate2Address } from '@ethersproject/address'

const token0 = '0xCAFE000000000000000000000000000000000000000000000000000000000000' // change me!
const token1 = '0xF00D00000000000000000000000000000000000000000000000000000000000' // change me!

const pair = getCreate2Address(
  FACTORY_ADDRESS,
  keccak256(['bytes'], [pack(['address', 'address'], [token0, token1])]),
  INIT_CODE_HASH
)
```

id: overview sidebar_position: 1 title: Overview

The Uniswap V2 SDK

Welcome to the Uniswap Protocol V2 SDK. To begin, we recommend looking at the [Guides](#) and for deeper reference see the [V2 SDK Github](#) repo.

Uniswap V2 SDK

- [V2 SDK Github](#)
- [SDK Core Github](#)
- [SDK NPM Package](#)

 Unit Tests passing  Lint passing npm@latest v3.0.1 minzipped size 47.1 kB

--- id: getting-started title:

Getting Started

The pages that follow contain technical reference information on the Uniswap SDK. Looking for a [quick start](#) instead? You may also want to jump into a [guide](#), which offers a friendlier introduction to the SDK!

The SDK is written in TypeScript, has a robust test suite, performs arbitrary precision arithmetic, and supports rounding to significant digits or fixed decimal places. The principal exports of the SDK are *entities*: classes that contain initialization and validation checks, necessary data fields, and helper functions.

An important concept in the SDK is *fractions*. Because Solidity performs integer math, care must be taken in non-EVM environments to faithfully replicate the actual computation carried out on-chain. The first concern here is to ensure that an overflow-safe integer implementation is used. Ideally, the SDK would be able to use native [BigInts](#). However, until support becomes more widespread, [JSBI](#) objects are used instead, with the idea that once Bigints proliferate, this dependency can be compiled away. The second concern is precision loss due to, for example, chained price ratio calculations. To address this issue, all math operations are performed as fraction operations, ensuring arbitrary precision up until the point that values are rounded for display purposes, or truncated to fit inside a fixed bit width.

The SDK works for all chains on which the [factory](#) is deployed.

Code

The [source code is available on GitHub](#).

Dependencies

The SDK declares its dependencies as [peer dependencies](#). This is for two reasons:

- prevent installation of unused dependencies (e.g. `@ethersproject/providers` and `@ethersproject/contracts`, only used in `Fetcher`)
- prevent duplicate `@ethersproject` dependencies with conflicting versions

However, this means you must install these dependencies alongside the SDK, if you do not already have them installed.

id: token title: Token

```
constructor(chainId: ChainId, address: string, decimals: number, symbol?: string, name?: string)
```

The Token entity represents an ERC-20 token at a specific address on a specific chain.

Example

```
import { ChainId, Token } from '@uniswap/sdk'

const token = new Token(ChainId.MAINNET, '0xc0FFee000000000000000000000000000000000000000000000000000000000', 18, 'HOT', 'Caffeine')
```

Properties

chainId

```
chainId: ChainId
```

See [ChainId](#)

address

```
address: string
```

decimals

```
decimals: number
```

symbol

```
symbol?: string
```

name

```
name?: string
```

Methods

equals

```
equals(other: Token): boolean
```

Checks if the current instance is equal to another (has an identical chainId and address).

sortsBefore

```
sortsBefore(other: Token): boolean
```

Checks if the current instance sorts before another, by address.

id: pair title: Pair

```
constructor(tokenAmountA: TokenAmount, tokenAmountB: TokenAmount)
```

The Pair entity represents a Uniswap pair with a balance of each of its pair tokens.

Example

```
import { ChainId, Token, TokenAmount, Pair } from '@uniswap/sdk'

const HOT = new Token(ChainId.MAINNET, '0xc0FFee000000000000000000000000000000000000000000000000000000000', 18, 'HOT', 'Caffeine')
const NOT = new Token(ChainId.MAINNET, '0xDeCAF0000000000000000000000000000000000000000000000000000000000', 18, 'NOT', 'Caffeine')

const pair = new Pair(new TokenAmount(HOT, '20000000000000000000'), new TokenAmount(NOT, '10000000000000000000'))
```

Static Methods

getAddress

```
getAddress(tokenA: Token, tokenB: Token): string
```

Computes the pair address for the passed [Tokens](#). See [Pair Addresses](#).

Properties

liquidityToken

```
liquidityToken: Token
```

A Token representing the liquidity token for the pair. See [Pair \(ERC-20\)](#).

token0

```
token0: Token
```

See [Token0](#).

token1

```
token1: Token
```

See [Token1](#).

reserve0

```
reserve0: TokenAmount
```

The reserve of token0.

reserve1

```
reserve1: TokenAmount
```

The reserve of token1.

Methods

reserveOf

```
reserveOf(token: Token): TokenAmount
```

Returns reserve0 or reserve1, depending on whether token0 or token1 is passed in.

getOutputAmount

```
getOutputAmount(inputAmount: TokenAmount): [TokenAmount, Pair]
```

Pricing function for exact input amounts. Returns maximum output amount based on current reserves and the new Pair that would exist if the trade were executed.

getInputAmount

```
getInputAmount(outputAmount: TokenAmount): [TokenAmount, Pair]
```

Pricing function for exact output amounts. Returns minimum input amount based on current reserves and the new Pair that would exist if the trade were executed.

getLiquidityMinted

```
getLiquidityMinted(totalSupply: TokenAmount, tokenAmountA: TokenAmount, tokenAmountB: TokenAmount): TokenAmount
```

Calculates the exact amount of liquidity tokens minted from a given amount of token0 and token1.

- totalSupply must be looked up on-chain.
- The value returned from this function *cannot* be used as an input to getLiquidityValue.

getLiquidityValue

```
getLiquidityValue(  
  token: Token,  
  totalSupply: TokenAmount,  
  liquidity: TokenAmount,  
  feeOn: boolean = false,  
  kLast?: BigintIsh  
) : TokenAmount
```

Calculates the exact amount of token0 or token1 that the given amount of liquidity tokens represent.

- totalSupply must be looked up on-chain.
- If the protocol charge is on, feeOn must be set to true, and kLast must be provided from an on-chain lookup.
- Values returned from this function *cannot* be used as inputs to getLiquidityMinted.

id: route title: Route

```
constructor(pairs: Pair[], input: Token)
```

The Route entity represents one or more ordered Uniswap pairs with a fully specified path from input token to output token.

Example

```
import { ChainId, Token, TokenAmount, Pair, Route } from '@uniswap/sdk'  
  
const HOT = new Token(ChainId.MAINNET, '0xc0FFee000000000000000000000000000000000000000000', 18, 'HOT', 'Caffeine')  
const NOT = new Token(ChainId.MAINNET, '0xDeCAF00000000000000000000000000000000000000000000000000000000000000', 18, 'NOT', 'Caffeine')  
const HOT_NOT = new Pair(new TokenAmount(HOT, '200000000000000000000000000000'), new TokenAmount(NOT, '100000000000000000000000000000'))  
  
const route = new Route([HOT_NOT], NOT)
```

Properties

pairs

```
pairs: Pair[]
```

The ordered pairs that the route is comprised of.

path

```
path: Token[]
```

The full path from input token to output token.

input

```
input: string
```

The input token.

output

```
output: string
```

The output token.

midPrice

```
midPrice: Price
```

Returns the current mid price along the route.

id: trade title: Trade

```
constructor(route: Route, amount: TokenAmount, tradeType: TradeType)
```

The Trade entity represents a fully specified trade along a route. This entity supplies all the information necessary to craft a router transaction.

Example

```
import { ChainId, Token, TokenAmount, Pair, Trade, TradeType, Route } from '@uniswap/sdk'

const HOT = new Token(ChainId.MAINNET, '0xc0FFee00000000000000000000000000000000000000000000', 18, 'HOT', 'Caffeine')
const NOT = new Token(ChainId.MAINNET, '0xDeCAF00000000000000000000000000000000000000000000000000000000000000', 18, 'NOT', 'Caffeine')
const HOT_NOT = new Pair(new TokenAmount(HOT, '20000000000000000000'), new TokenAmount(NOT, '10000000000000000000'))
const NOT_TO_HOT = new Route([HOT_NOT], NOT)

const trade = new Trade(NOT_TO_HOT, new TokenAmount(NOT, '1000000000000000'), TradeType.EXACT_INPUT)
```

Properties

route

```
route: Route
```

The `path` property of the route should be passed as the path parameter to router functions.

tradeType

```
tradeType: TradeType
```

`TradeType.EXACT_INPUT` corresponds to `swapExact*For*` router functions. `TradeType.EXACT_OUTPUT` corresponds to `swap*ForExact*` router functions.

inputAmount

```
inputAmount: TokenAmount
```

For exact input trades, this value should be passed as `amountIn` to router functions. For exact output trades, this value should be multiplied by a factor >1, representing slippage tolerance, and passed as `amountInMax` to router functions.

outputAmount

```
outputAmount: TokenAmount
```

For exact output trades, this value should be passed as `amountOut` to router functions. For exact input trades, this value should be multiplied by a factor <1, representing slippage tolerance, and passed as `amountOutMin` to router functions.

executionPrice

```
executionPrice: Price
```

The average price that the trade would execute at.

nextMidPrice

```
nextMidPrice: Price
```

What the new mid price would be if the trade were to execute.

slippage

```
slippage: Percent
```

The slippage incurred by the trade.

- Strictly > .30%.

Methods

In the context of the following two methods, slippage refers to the percent difference between the actual price and the trade `executionPrice`.

minimumAmountOut (since 2.0.4)

```
minimumAmountOut(slippageTolerance: Percent): TokenAmount
```

Returns the minimum amount of the output token that should be received from a trade, given the slippage tolerance.

Useful when constructing a transaction for a trade of type `EXACT_INPUT`.

maximumAmountIn (since 2.0.4)

```
maximumAmountIn(slippageTolerance: Percent): TokenAmount
```

Returns the maximum amount of the input token that should be spent on the trade, given the slippage tolerance.

Useful when constructing a transaction for a trade of type `EXACT_OUTPUT`.

Static methods

These static methods provide ways to construct ideal trades from lists of pairs. Note these methods do not perform any aggregation across routes, as routes are linear. It's possible that a better price can be had by combining multiple trades across different routes.

bestTradeExactIn

Given a list of pairs, a fixed amount in, and token amount out, this method returns the best `maxNumResults` trades that swap an input token amount to an output token, making at most `maxHops` hops. The returned trades are sorted by output amount, in decreasing order, and all share the given input amount.

```
Trade.bestTradeExactIn(  
    pairs: Pair[],  
    amountIn: TokenAmount,  
    tokenOut: Token,  
    { maxNumResults = 3, maxHops = 3 }: BestTradeOptions = {}): Trade[]
```

bestTradeExactOut

Similar to the above method, but targets a fixed output token amount. The returned trades are sorted by input amount, in increasing order, and all share the given output amount.

```
Trade.bestTradeExactOut(  
    pairs: Pair[],  
    tokenIn: Token,  
    amountOut: TokenAmount,  
    { maxNumResults = 3, maxHops = 3 }: BestTradeOptions = {}): Trade[]
```

id: fractions title: Fractions

Fraction

```
constructor(numerator: BigintIsh, denominator: BigintIsh = ONE)
```

The base class which all subsequent fraction classes extend. **Not meant to be used directly.**

Properties

numerator

```
numerator: JSBI
```

denominator

```
denominator: JSBI
```

quotient

```
quotient: JSBI
```

Performs floor division.

Methods

invert

```
invert(): Fraction
```

add

```
add(other: Fraction | BigintIsh): Fraction
```

subtract

```
subtract(other: Fraction | BigintIsh): Fraction
```

multiply

```
multiply(other: Fraction | BigintIsh): Fraction
```

divide

```
divide(other: Fraction | BigintIsh): Fraction
```

toSignificant

```
toSignificant(  
    significantDigits: number,  
    format: object = { groupSeparator: '' },  
    rounding: Rounding = Rounding.ROUND_HALF_UP  
)
```

Formats a fraction to the specified number of significant digits.

- For format options, see [toFormat](#).

toFixed

```
toFixed(  
    decimalPlaces: number,  
    format: object = { groupSeparator: '' },  
    rounding: Rounding = Rounding.ROUND_HALF_UP  
)
```

Formats a fraction to the specified number of decimal places.

- For format options, see [toFormat](#).

Percent

Responsible for formatting percentages (10% instead of 0.1).

Example

```
import { Percent } from '@uniswap/sdk'  
  
const percent = new Percent('60', '100')  
console.log(percent.toSignificant(2)) // 60
```

toSignificant

See [toSignificant](#).

toFixed

See [toFixed](#).

TokenAmount

```
constructor(token: Token, amount: BignintIsh)
```

Responsible for formatting token amounts with specific decimal places.

Example

```
import { Token, TokenAmount } from '@uniswap/sdk'

const FRIED = new Token(ChainId.MAINNET, '0xfalaFe1000000000000000000000000000000000000000000000')

const tokenAmount = new TokenAmount(FRIED, '300000000000000000')
console.log(tokenAmount.toExact()) // 3
```

Properties

token

```
token: Token
```

raw

```
raw: JSBI
```

Returns the full token amount, unadjusted for decimals.

Methods

add

```
add(other: TokenAmount): TokenAmount
```

subtract

```
subtract(other: TokenAmount): TokenAmount
```

toSignificant

See [toSignificant](#).

toFixed

See [toFixed](#).

toExact

```
toExact(format: object = { groupSeparator: '' }): string
```

Price

```
constructor(baseToken: Token, quoteToken: Token, denominator: BignintIsh, numerator: BignintIsh)
```

Responsible for denominating the relative price between two tokens. Denominator and numerator must be unadjusted for decimals.

Example

```
import { ChainId, WETH as WETHs, Token, Price } from '@uniswap/sdk'

const WETH = WETHs[ChainId.MAINNET]
const ABC = new Token(ChainId.MAINNET, '0xabc0000000000000000000000000000000000000000000000000000000', 18, 'ABC')

const price = new Price(WETH, ABC, '10000000000000000000', '12300000000000000000')
console.log(price.toSignificant(3)) // 123
```

This example shows the ETH/XYZ price, where ETH is the base token, and XYZ is the quote token. The price is constructed from an amount of XYZ (the numerator) / an amount of WETH (the denominator).

Static Methods

fromRoute

```
fromRoute(route: Route): Price
```

Properties

baseToken

```
baseToken: Token
```

quoteToken

```
quoteToken: Token
```

scalar

```
scalar: Fraction
```

Used to adjust the price for the decimals of the base and quote tokens.

raw

```
raw: Fraction
```

Returns the raw price, unadjusted for decimals.

adjusted

```
adjusted: Fraction
```

Returns the price, adjusted for decimals.

Methods

invert

```
invert(): Price
```

multiply

```
multiply(other: Price): Price
```

quote

```
quote(tokenAmount: TokenAmount): TokenAmount
```

Given an asset amount, returns an equivalent value of the other asset, according to the current price.

toSignificant

See [toSignificant](#).

toFixed

See [toFixed](#).

id: fetcher title: Fetcher

The data fetching logic is split from the rest of the code for better tree-shaking, i.e. so that it does not get packaged into your code unless it is used. The SDK is otherwise unconcerned with how you get data from the blockchain.

This class contains static methods for constructing instances of pairs and tokens from on-chain data. It cannot be constructed.

Static Methods

fetchTokenData

```
async fetchTokenData(  
  chainId: ChainId,  
  address: string,  
  provider = getDefaultProvider(getNetwork(chainId)),  
  symbol?: string,  
  name?: string  
): Promise<Token>
```

Initializes a class instance from a chainId and token address, if the decimals of the token are unknown and cannot be fetched externally. Decimals are fetched via an [ethers.js](#) v5 provider. If not passed in, a default provider is used.

fetchPairData

```
async fetchPairData(  
  tokenA: Token,  
  tokenB: Token,  
  provider = getDefaultProvider(getNetwork(tokenA.chainId))  
): Promise<Pair>
```

Initializes a class instance from two Tokens, if the pair's balances of these tokens are unknown and cannot be fetched externally. Pair reserves are fetched via an [ethers.js](#) v5 provider. If not passed in, a default provider is used.

id: other-exports title: Other Exports

JSBI

```
import { JSBI } from '@uniswap/sdk'  
// import JSBI from 'jsbi'
```

The default export from [jsbi](#).

BigintIsh

```
import { BigintIsh } from '@uniswap/sdk'  
// type BigintIsh = JSBI | bigint | string
```

A union type comprised of all types that can be cast to a JSBI instance.

ChainId

```
import { ChainId } from '@uniswap/sdk'  
// enum ChainId {  
//   MAINNET = 1,  
//   ROPSTEN = 3,  
//   RINKEBY = 4,  
//   GÖRLI = 5,  
//   KOVAN = 42  
// }
```

A enum denominating supported chain IDs.

TradeType

```
import { TradeType } from '@uniswap/sdk'  
// enum TradeType {  
//   EXACT_INPUT,  
//   EXACT_OUTPUT  
// }
```

A enum denominating supported trade types.

Rounding

```
import { Rounding } from '@uniswap/sdk'  
// enum Rounding {  
//   ROUND_DOWN,  
//   ROUND_HALF_UP,  
//   ROUND_UP  
// }
```

A enum denominating supported rounding options.

FACTORY_ADDRESS

```
import { FACTORY_ADDRESS } from '@uniswap/sdk'
```

The [factory address](#).

INIT_CODE_HASH

```
import { INIT_CODE_HASH } from '@uniswap/sdk'
```

See [pair addresses](#).

MINIMUM_LIQUIDITY

```
import { MINIMUM_LIQUIDITY } from '@uniswap/sdk'
```

See [minimum liquidity](#).

InsufficientReservesError

```
import { InsufficientReservesError } from '@uniswap/sdk'
```

InsufficientInputAmountError

```
import { InsufficientInputAmountError } from '@uniswap/sdk'
```

WETH

```
import { WETH } from '@uniswap/sdk'
```

An object whose values are [WETH Token](#) instances, indexed by [ChainId](#).

id: background title: Background

Before integrating with Uniswap, it may be helpful for newcomers to review the following background information on some important developer web3 concepts, the structure of our examples, and SDK concepts.

:::info Already familiar with web3 development and/or the basics of our SDK and want to get right to the code? Start with our first guide, [Getting a Quote!](#) :::

Providers

Communication with the blockchain is typically done through a provider and local models of smart contracts and their [ABIs](#).

To achieve this, our examples use the [ethers.js](#) library. To instantiate a provider you will need a data source. Our examples offer two options:

- **JSON RPC URL:** If you are working directly with the Ethereum mainnet or a local fork, products such as [infura](#) offer JSON RPC URLs for a wide variety of chains and testnets. For our examples, we'll only be using the Ethereum mainnet.
- **Wallet Extension:** If you are connecting to a wallet browser extension, these wallets embed a source directly into the Javascript window object as `window.ethereum`. This object surfaces information about the user's wallets and provides the ability to communicate with the connected chain. Importantly for our examples, it can be used with `ethers.js` to construct a provider.

Uniswap's Runnable Examples

Each guide is accompanied and driven by [runnable examples](#) using React to provide a basic UI for interacting with the example. Each examples provides relevant options such as running against a local blockchain or connecting to the Ethereum mainnet directly. You also have the option of using a wallet extension which can be connected to either environment.

Inputs and environment settings are configured in each example's `config.ts` and allows for simple setup and configuration.

Developing and Testing

To test your code, we recommend utilizing a local fork of the Ethereum mainnet. To help facilitate easy testing, each example includes a quickstart for running the local chain with a test wallet. To further test, we also recommend using a wallet extension and connecting to the local chain. Finally, each example can be run against the Ethereum mainnet if desired. Full development instructs can be found in the `README.md` of each code example.

Utility Libraries

Each example is concentrated into a single file within the `libs/` folder of the example, with the entry points noted in each guide and README.

To allow the guides to focus on the SDK's core functionality, additional basic building blocks can be found in each example's `libs` folder. The exported functionality from these files is intended to be the minimum needed for each example and not a complete library for production usage. These also include storing core constants such as definitions for tokens, ABI's, and blockchain addresses that can distract from the core concepts. Below are summaries of the helping libraries you will encounter.

Provider Utilities

`provider.ts` wraps the basics of `ethers.js` and connecting to wallet extensions into an abstracted view of a provider, a wallet address, and the ability to send transactions. It also helps abstract the configured environment you wish to run against in your example without making code changes outside of your configuration.

Wallet Utilities

`wallet.ts` offers the ability to query a wallet (whether connected via an extension or defined in code/config) for its balances and other essential information.

Pool Information

`pool.ts` contains the basic querying of pool information when not essential / core to the relevant guide

Display Utilities

`conversion.ts` provides display and light math wrappers to help show human readable prices when dealing with currency amounts (typically stored as raw numbers and the decimal placement separate for precision reasons) in the form of two functions: `fromReadableAmount` and `toReadableAmount`

Notable SDK Structures and Concepts

When working with the SDK it can be helpful to understand some of the design choices and why they are needed. Below you can find a few important concepts.

ABI's

To allow others to interact with a smart contract, each contract exposes an ABI (Application Binary Interface). As these are defined on the blockchain, we must ensure the correct definitions are provided to our Javascript functions. ABI's are provided from various SDK's and imported in as needed. Some examples will define an ABI directly as needed.

CurrencyAmount and JSBI

Cryptocurrency applications often work with very small fractions of tokens. As a result, high precision is very important. To ensure precision is upheld, the `CurrencyAmount` class helps store exact values as fractions and utilizes [JSBI](#) for compatibility across the web. To display these amounts nicely to users, additional work is sometimes required.

Currency

The `Currency` class can represent both native currency (ETH) and an ERC20 Token .

Currencies vary in their relative value, so the `Token` class allows your application to define the number of decimals needed for each currency along with the currency's address, symbol, and name.

id: quoting title: Getting a Quote

Introduction

This guide will cover how to get the current quotes for any token pair on the Uniswap protocol. It is based on the [Quoting code example](#), found in the Uniswap code examples [repository](#). To run this example, check out the examples's [README](#) and follow the setup instructions.

:::info If you need a briefer on the SDK and to learn more about how these guides connect to the examples repository, please visit our [background page!](#) :::

In this example we will use `quoteExactInputSingle` to get a quote for the pair **USDC - WETH**. The inputs are the **token in**, the **token out**, the **amount in** and the **fee**.

The **fee** input parameters represents the swap fee that distributed to all in range liquidity at the time of the swap. It is one of the identifiers of a Pool, the others being **tokenIn** and **tokenOut**.

The guide will cover:

1. Computing the Pool's deployment address
2. Referencing the Pool contract and fetching metadata
3. Referencing the Quoter contract and getting a quote

At the end of the guide, we should be able to fetch a quote for the given input token pair and the input token amount with the press of a button on the web application.

For this guide, the following Uniswap packages are used:

- [@uniswap/v3-sdk](#)
- [@uniswap/sdk-core](#)

The core code of this guide can be found in [quote.ts](#)

Computing the Pool's deployment address

To interact with the **USDC - WETH** Pool contract, we first need to compute its deployment address. The SDK provides a utility method for that:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-
sdk/quoting/src/libs/quote.ts#L40-L45
```

Since each *Uniswap V3 Pool* is uniquely identified by 3 characteristics (token in, token out, fee), we use those in combination with the address of the *PoolFactory* contract to compute the address of the **USDC - ETH Pool**. These parameters have already been defined in our configuration file:

```
https://github.com/Uniswap/examples/blob/1ef393c2b8f8206a3dc5a42562382c267bcc361b/v3-sdk/quoting/src/config.ts#L34-L39
```

Referencing the Pool contract and fetching metadata

Now that we have the deployment address of the **USDC - ETH Pool**, we can construct an instance of an `ethers.Contract` to interact with it:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/quoting/src/libs/quote.ts#L47-L51
```

To construct the `Contract` we need to provide the address of the contract, its ABI and the provider that will carry out the RPC call for us. We get access to the contract's ABI through the [@uniswap/v3-core](#) package, which holds the core smart contracts of the Uniswap V3 protocol:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/quoting/src/libs/quote.ts#L5
```

Having constructed our reference to the contract, we can now access its methods through our provider. We use a batch `Promise` call. This approach queries state data concurrently, rather than sequentially, to avoid out of sync data that may be returned if sequential queries are executed over the span of two blocks:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/quoting/src/libs/quote.ts#L52-L56
```

The return values of these methods will become inputs to the quote fetching function.

:::note In this example, the metadata we fetch is already present in our inputs. This guide fetches this information first in order to show how to fetch any metadata, which will be expanded on in future guides. :::

Referencing the Quoter contract and getting a quote

Like we did for the Pool contract, we need to construct an instance of an `ethers.Contract` for our Quoter contract in order to interact with it:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/quoting/src/libs/quote.ts#L14-L18
```

We get access to the contract's ABI through the [@uniswap/v3-periphery](#) package, which holds the periphery smart contracts of the Uniswap V3 protocol:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/quoting/src/libs/quote.ts#L4
```

We can now use our Quoter contract to obtain the quote.

In an ideal world, the quoter functions would be `view` functions, which would make them very easy to query on-chain with minimal gas costs. However, the Uniswap V3 Quoter contracts rely on state-changing calls designed to be reverted to return the desired data. This means calling the quoter will be very expensive and should not be called on-chain.

To get around this difficulty, we can use the `callStatic` method provided by the `ethers.js Contract` instances. This is a useful method that submits a state-changing transaction to an Ethereum node, but asks the node to simulate the state change, rather than to execute it. Our script can then return the result of the simulated state change:

```
https://github.com/Uniswap/examples/blob/2e8fb5ef56e502d4eb0261e4abff262c33a30760/v3-sdk/quoting/src/libs/quote.ts#L21-L30
```

The result of the call is the number of output tokens you'd receive for the quoted swap.

It should be noted that `quoteExactInputSingle` is only 1 of 4 different methods that the quoter offers:

1. `quoteExactInputSingle` - given the amount you want to swap, produces a quote for the amount out for a swap of a single pool
2. `quoteExactInput` - given the amount you want to swap, produces a quote for the amount out for a swap over multiple pools
3. `quoteExactOutputSingle` - given the amount you want to get out, produces a quote for the amount in for a swap over a single pool
4. `quoteExactOutput` - given the amount you want to get out, produces a quote for the amount in for a swap over multiple pools

Next Steps

Now that you're able to make a quote, check out our next guide on [trading](#) using this quote!

id: trading title: Executing a Trade

Introduction

This guide will build off our [quoting guide](#) and show how to use a quote to construct and execute a trade on the Uniswap V3 protocol. It is based on the [Trading code example](#), found in the Uniswap code examples [repository](#). To run this example, check out the guide's [README](#) and follow the setup instructions.

:::info If you need a briefer on the SDK and to learn more about how these guides connect to the examples repository, please visit our [background](#) page! :::

In this example we will trade between two ERC20 tokens: **WETH and USDC**. The tokens, amount of input token, and the fee level can be configured as inputs.

The guide will **cover**:

1. Constructing a route from pool information
2. Constructing an unchecked trade
3. Executing a trade

At the end of the guide, we should be able to create and execute a trade between any two ERC20 tokens using the example's included UI.

:::note Included in the example application is functionality to wrap/unwrap ETH as needed to fund the example `WETH` to `USDC` swap directly from an `ETH` balance. :::

For this guide, the following Uniswap packages are used:

- [@uniswap/v3-sdk](#)
- [@uniswap/sdk-core](#)

The core code of this guide can be found in [trading.ts](#)

Constructing a route from pool information

To construct our trade, we will first create a model instance of a `Pool`. We will first extract the needed metadata from the relevant pool contract. Metadata includes both constant information about the pool as well as information about its current state stored in its first slot:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-sdk/trading/src/libs/pool.ts#L38-L56
```

Using this metadata along with our inputs, we will then construct a `Pool`:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-sdk/trading/src/libs/trading.ts#L41-L50
```

With this `Pool`, we can now construct a route to use in our trade. We will reuse our previous quoting code to calculate the output amount we expect from our trade:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-sdk/trading/src/libs/trading.ts#L52-L56
```

Constructing an unchecked trade

Once we have constructed the route object, we now need to obtain a quote for the given `inputAmount` of the example:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-sdk/trading/src/libs/trading.ts#L58
```

As shown below, the quote is obtained using the `v3-sdk`'s `SwapQuoter`, in contrast to the [previous quoting guide](#), where we directly accessed the smart contact:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-sdk/trading/src/libs/trading.ts#L128-L141
```

The `SwapQuoter`'s `quoteCallParameters` function, gives us the calldata needed to make the call to the `Quoter`, and we then decode the returned quote:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-
sdk/trading/src/libs/trading.ts#L143-L148
```

With the quote and the route, we can now construct an unchecked trade using the route in addition to the output amount from a quote based on our input:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-
sdk/trading/src/libs/trading.ts#L60-L74
```

This example uses an exact input trade, but we can also construct a trade using exact output assuming we adapt our quoting code accordingly.

Executing a trade

Once we have created a trade, we can now execute this trade with our provider. First, we must give the `SwapRouter` approval to spend our tokens for us:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-
sdk/trading/src/libs/trading.ts#L90
```

Then, we set our options that define how much time and slippage can occur in our execution as well as the address to use for our wallet:

```
https://github.com/Uniswap/examples/blob/c4667fadbf13584268bbee2e0e0f556558a474751/v3-
sdk/trading/src/libs/trading.ts#L97-L101
```

Next, we use the Uniswap `SwapRouter` to get the associated call parameters for our trade and options:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-
sdk/trading/src/libs/trading.ts#L103
```

Finally, we can construct a transaction from the method parameters and send the transaction:

```
https://github.com/Uniswap/examples/blob/bbee4b974768ff1668ac56e27d1fe840060bb61b/v3-
sdk/trading/src/libs/trading.ts#L105-L114
```

Next Steps

The resulting example allows for trading between any two ERC20 tokens, but this can be suboptimal for the best pricing and fees. To achieve the best possible price, we use the Uniswap auto router to route through pools to get an optimal cost. Our [routing](#) guide will show you how to use this router and execute optimal swaps.

id: routing title: Routing a Swap

Introduction

This guide will cover how to use Uniswap's smart order router to compute optimal routes and execute swaps. Rather than trading between a single pool, smart routing may use multiple hops (as many as needed) to ensure that the end result of the swap is the optimal price. It is based on the [routing code example](#), found in the Uniswap code examples [repository](#). To run this example, check out the guide's [README](#) and follow the setup instructions.

:::info If you need a briefer on the SDK and to learn more about how these guides connect to the examples repository, please visit our [background](#) page! :::

In this example we will trade between **WETH** and **USDC**, but you can configure your example to use any two currencies and amount of input currency.

The guide will **cover**:

1. Creating a router instance
2. Creating a route
3. Swapping using a route

At the end of the guide, we should be able to create a route and execute a swap between any two currencies tokens using the example's included UI.

For this guide, the following Uniswap packages are used:

- [@uniswap/v3-sdk](#)
- [@uniswap/sdk-core](#)
- [@uniswap/smart-order-router](#)

The core code of this guide can be found in [routing.ts](#)

Creating a router instance

To compute our route, we will use the `@uniswap/smart-order-router` package, specifically the `AlphaRouter` class which requires a `chainId` and a `provider`. Note that routing is not supported for local forks, so we will use a mainnet provider even when swapping on a local fork:

```
https://github.com/Uniswap/examples/blob/38ff60aeb3ad8ff839db9e7952a726ca7d6b68fd/v3-sdk/routing/src/libs/routing.ts#L24-L27
```

Creating a route

Next, we will create our options conforming to the `SwapOptionsSwapRouter02` interface, defining the wallet to use, slippage tolerance, and deadline for the transaction:

```
https://github.com/Uniswap/examples/blob/c4667fadbf13584268bbbe2e0e0f556558a474751/v3-sdk/routing/src/libs/routing.ts#L33-L38
```

Using these options, we can now create a trade (`TradeType.EXACT_INPUT` or `TradeType.EXACT_OUTPUT`) with the currency and the input amount to use to get a quote. For this example, we'll use an `EXACT_INPUT` trade to get a quote outputted in the quote currency.

```
https://github.com/Uniswap/examples/blob/38ff60aeb3ad8ff839db9e7952a726ca7d6b68fd/v3-sdk/routing/src/libs/routing.ts#L36-L47
```

Swapping using a route

First, we need to give approval to the `SwapRouter` smart contract to spend our tokens for us:

```
https://github.com/Uniswap/examples/blob/0071bb5883fba6f4cc39a5f1644ac941e4f24822/v3-sdk/routing/src/libs/routing.ts#L66
```

Once the approval has been granted and using the route, we can now execute the trade using the route's computed calldata, values, and gas values:

```
https://github.com/Uniswap/examples/blob/38ff60aeb3ad8ff839db9e7952a726ca7d6b68fd/v3-sdk/routing/src/libs/routing.ts#L61-L68
```

After swapping, you should see the currency balances update in the UI shortly after the block is confirmed.

Next Steps

Now that you're familiar with trading, consider checking out our next guides on [pooling liquidity to Uniswap!](#)

id: minting title: Minting a Position

Introduction

This guide will cover how to create (or mint) a liquidity position on the Uniswap V3 protocol. It is based on the [minting a position code example](#), found in the Uniswap code examples [repository](#). To run this example, check out the example's [README](#) and follow the setup instructions.

:::info If you need a briefer on the SDK and to learn more about how these guides connect to the examples repository, please visit our [background page!](#) :::

In the Uniswap V3 protocol, liquidity positions are represented using non-fungible tokens. In this guide we will use the `NonfungiblePositionManager` class to help us mint a liquidity position for the **USDC - DAI** pair. The inputs to our guide are the **two tokens**

that we are pooling for, the **amount** of each token we are pooling for and the **Pool fee**.

The guide will **cover**:

1. Giving approval to transfer our tokens
2. Creating an instance of a `Pool`
3. Calculating our `Position` from our input tokens
4. Configuring and executing our minting transaction

At the end of the guide, given the inputs above, we should be able to mint a liquidity position with the press of a button and view the position on the UI of the web application.

For this guide, the following Uniswap packages are used:

- [@uniswap/v3-sdk](#)
- [@uniswap/sdk-core](#)
- [@uniswap/smart-order-router](#)

The core code of this guide can be found in [mintPosition\(\)](#)

Giving approval to transfer our tokens

The first step is to give approval to the protocol's `NonfungiblePositionManager` to transfer our tokens:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/minting-
position/src/libs/positions.ts#L46-L51
```

The logic to achieve that is wrapped in the `getTokenTransferApprovals` function. In short, since both **USDC** and **DAI** are ERC20 tokens, we setup a reference to their smart contracts and call the `approve` function:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/minting-
position/src/libs/positions.ts#L202-L211
```

Creating an instance of a `Pool`

Having approved the transfer of our tokens, we now need to get data about the pool for which we will provide liquidity, in order to instantiate a `Pool` class.

To start, we compute our `Pool`'s address by using a helper function and passing in the unique identifiers of a `Pool` - the **two tokens** and the `Pool fee`. The `fee` input parameter represents the swap fee that is distributed to all in range liquidity at the time of the swap:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/minting-
position/src/libs/pool.ts#L24-L29
```

Then, we get the `Pool`'s data by creating a reference to the `Pool`'s smart contract and accessing its methods:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/minting-
position/src/libs/pool.ts#L31-L45
```

Having collected the required data, we can now create an instance of the `Pool` class:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/minting-
position/src/libs/positions.ts#L111-L118
```

Calculating our `Position` from our input tokens

Having created the instance of the `Pool` class, we can now use that to create an instance of a `Position` class, which represents the price range for a specific pool that LPs choose to provide in:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/minting-
position/src/libs/positions.ts#L121-L132
```

We use the `fromAmounts` static function of the `Position` class to create an instance of it, which uses the following parameters:

- The `tickLower` and `tickUpper` parameters specify the price range at which to provide liquidity. This example calls `nearestUsableTick` to get the current useable tick and adjust the lower parameter to be below it by two `tickSpacing` and the upper to be above it by two `tickSpacing`. This guarantees that the provided liquidity is "in range", meaning it will be earning fees upon minting this position

- **amount0** and **amount1** define the maximum amount of currency the liquidity position can use. In this example, we supply these from our configuration parameters.

Given those parameters, `fromAmounts` will attempt to calculate the maximum amount of liquidity we can supply.

Configuring and executing our minting transaction

The Position instance is then passed as input to the `NonfungiblePositionManager`'s `addCallParameters` function. The function also requires an `AddLiquidityOptions` object as its second parameter. This is either of type `MintOptions` for minting a new position or `IncreaseOptions` for adding liquidity to an existing position. For this example, we're using a `MintOptions` to create our position.

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/minting-position/src/libs/positions.ts#L78-L88
```

The function returns the calldata as well as the value required to execute the transaction:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/minting-position/src/libs/positions.ts#L91-L100
```

The effect of the transaction is to mint a new Position NFT. We should see a new position with liquidity in our list of positions.

Next Steps

Once you have minted a position, our next guide ([Adding and Removing Liquidity](#)) will demonstrate how you can add and remove liquidity from that minted position!

id: modifying-position title: Adding & Removing Liquidity

Introduction

This guide will cover how to modify a liquidity position by adding or removing liquidity on the Uniswap V3 protocol. It is based on the [modifying_a_position_code_example](#), found in the Uniswap code examples [repository](#). To run this example, check out the examples's [README](#) and follow the setup instructions.

:::info If you need a briefer on the SDK and to learn more about how these guides connect to the examples repository, please visit our [background page!](#) :::

In the Uniswap V3 protocol, liquidity positions are represented using non-fungible tokens. In this guide we will use the `NonfungiblePositionManager` class to help us mint a liquidity position and then modify the provided liquidity for the **USDC - DAI** pair. The inputs to our guide are the **two tokens** that we are pooling for, the **amount** of each token we are pooling for, the Pool **fee** and the **fraction** by which to **add and remove** from our position.

The guide will **cover**:

1. Adding liquidity to our position
2. Removing liquidity from our position

At the end of the guide, given the inputs above, we should be able to add or remove liquidity from a minted position with the press of a button and see the change reflected in our position and the balance of our tokens.

For this guide, the following Uniswap packages are used:

- [@uniswap/v3-sdk](#)
- [@uniswap/sdk-core](#)

The core code of this guide can be found in [addLiquidity\(\)](#) and [removeLiquidity\(\)](#)

:::note This guide assumes you are familiar with our [Minting a Position](#) guide. A minted position is required to add or remove liquidity from, so the buttons will be disabled until a position is minted.

Also note that we do not need to give approval to the `NonfungiblePositionManager` to transfer our tokens as we will have already done that when minting our position. :::

Adding liquidity to our position

Assuming we have already minted a position, our first step is to construct the modified position using our original position to calculate the amount by which we want to increase our current position:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-position/src/libs/positions.ts#L111-L125
```

```
position/src/libs/liquidity.ts#L46-L61
```

The function receives two arguments, which are the `CurrencyAmount`'s that are used to construct the `Position` instance. In this example, both of the arguments follow the same logic: we multiply the parameterized `tokenAmount` by the parameterized `fractionToAdd` since the new liquidity position will be added on top of the already minted liquidity position.

We then need to construct an options object of type [`AddLiquidityOptions`](#) similar to how we did in the minting case. In this case, we will use [`IncreaseOptions`](#):

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-
position/src/libs/liquidity.ts#L63-L67
```

Compared to minting, we have omitted the `recipient` parameter and instead passed in the `tokenId` of the position we previously minted.

The newly created position along with the options object are then passed to the `NonfungiblePositionManager`'s `addCallParameters`:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-
position/src/libs/liquidity.ts#L70-L73
```

The return values of `addCallParameters` are the calldata and value of the transaction we need to submit to increase our position's liquidity. We can now build and execute the transaction:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-
position/src/libs/liquidity.ts#L76-L85
```

After pressing the button, note how the balance of USDC and DAI drops and our position's liquidity increases.

Removing liquidity from our position

The `removeLiquidity` function is the mirror action of adding liquidity and will be somewhat similar as a result, requiring a position to already be minted.

To start, we create a position identical to the one we minted:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-
position/src/libs/liquidity.ts#L97-L112
```

We then need to construct an options object of type [`RemoveLiquidityOptions`](#):

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-
position/src/libs/liquidity.ts#L126-L133
```

Just as with adding liquidity, we have omitted the `recipient` parameter and instead passed in the `tokenId` of the position we previously minted.

We have also provided two additional parameters:

- `liquidityPercentage` determines how much liquidity is removed from our initial position (as a `Percentage`), and transfers the removed liquidity back to our address. We set this percentage from our guide configuration ranging from 0 (0%) to 1 (100%).
- `collectOptions` gives us the option to collect the fees, if any, that we have accrued for this position. In this example, we won't collect any fees, so we provide zero values. If you'd like to see how to collect fees without modifying your position, check out our [collecting fees](#) guide!

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-
position/src/libs/liquidity.ts#L114-L124
```

The position object along with the options object is passed to the `NonfungiblePositionManager`'s `removeCallParameters`, similar to how we did in the adding liquidity case:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-
position/src/libs/liquidity.ts#L135-L138
```

The return values `removeCallParameters` are the calldata and value that are needed to construct the transaction to remove liquidity from our position. We can build the transaction and send it for execution:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/modifying-position/src/libs/liquidity.ts#L141-L150
```

After pressing the button, note how the balance of USDC and DAI increases and our position's liquidity drops.

Next Steps

Now that you can mint and modify a position, check out how to [collect fees](#) from the position!

id: liquidity-fees title: Collecting Fees

Introduction

This guide will cover how to collect fees from a liquidity position on the Uniswap V3 protocol. It is based on the [collecting fees code example](#), found in the Uniswap code examples [repository](#). To run this example, check out the examples's [README](#) and follow the setup instructions.

:::info If you need a briefer on the SDK and to learn more about how these guides connect to the examples repository, please visit our [background page!](#) :::

In the Uniswap V3 protocol, liquidity positions are represented using non-fungible tokens. In this guide we will use the `NonfungiblePositionManager` class to help us mint a liquidity position for the **USDC - DAI** pair. We will then attempt to collect any fees that the position has accrued from those trading against our provisioned liquidity. The inputs to our guide are the **two tokens** that we are pooling for, the **amount** of each token we are pooling for, the Pool **fee** and the **max amount of accrued fees** we want to collect for each token.

The guide will **cover**:

1. Setting up our fee collection
2. Submitting our fee collection transaction

At the end of the guide, given the inputs above, we should be able to collect the accrued fees (if any) of a minted position with the press of a button and see the change reflected in our position and the balance of our tokens.

For this guide, the following Uniswap packages are used:

- [@uniswap/v3-sdk](#)
- [@uniswap/sdk-core](#)

The core code of this guide can be found in [collectFees\(\)](#).

:::note This guide assumes you are familiar with our [Minting a Position](#) guide. A minted position is required to add or remove liquidity from, so the buttons will be disabled until a position is minted.

Also note that we do not need to give approval to the `NonfungiblePositionManager` to transfer our tokens as we will have already done that when minting our position. :::

Setting up our fee collection

All of the fee collecting logic can be found in the `collectFees` function. Notice how the **Collect Fees** button is disabled until a position is minted. This happens because there will be no fees to collect unless there is a position whose liquidity has been traded against.

To start, we construct an options object of type `CollectOptions` that holds the data about the fees we want to collect:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/collecting-fees/src/libs/liquidity.ts#L44-L61
```

Similar to the other functions exposed by the `NonfungiblePositionManager`, we pass the `tokenId` and the `recipient` of the fees, which in this case is our function's input position id and our wallet's address.

The other two `CurrencyAmount` parameters (`expectedCurrencyOwed0` and `expectedCurrencyOwed1`) define the **maximum** amount of currency we expect to get collect through accrued fees of each token in the pool. We set these through our guide's configuration.

Submitting our fee collection transaction

We then get the call parameters for collecting our fees from our `NonfungiblePositionManager` using the constructed `CollectOptions`:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/collecting-fees/src/libs/liquidity.ts#L64-L65
```

The function above returns the calldata and value required to construct the transaction for collecting accrued fees. Now that we have both the calldata and value we needed for the transaction, we can build and execute the it:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/collecting-fees/src/libs/liquidity.ts#L68-L77
```

After pressing the button, if someone has traded against our position, we should be able to note how the balance of USDC and DAI increases as we collect fees.

Next Steps

The previous guides detail all the atomic steps needed to create and manage positions. However, these approaches may not use all of your desired currency. To ensure you are using your full funds while minimizing gas prices, check out our guide on [Swapping and Adding Liquidity](#) in a single transaction!

id: swap-and-add title: Swapping and Adding Liquidity

Introduction

This guide will cover how to execute a swap-and-add operation in a single atomic transaction. It is based on the [swap-and-add example](#), found in the Uniswap code examples [repository](#). To run this example, check out the examples's [README](#) and follow the setup instructions.

:::info If you need a briefer on the SDK and to learn more about how these guides connect to the examples repository, please visit our [background page!](#) :::

When adding liquidity to a Uniswap v3 pool, you must provide two assets in a particular ratio. In many cases, your contract or the user's wallet hold a different ratio of those two assets. In order to deposit 100% of your assets, you must first swap your assets to the optimal ratio and then add liquidity.

However, the swap may shift the balance of the pool and thus change the optimal ratio. To avoid that, we can execute this swap-and-add liquidity operation in an atomic fashion, using a router. The inputs to our guide are the **two tokens** that we are pooling for, the **amount** of each token we are pooling for, the **amount** of each token to swap-and-add, and the Pool **fee**.

The guide will **cover**:

1. Setup a router instance
2. Configuring our ratio calculation
3. Calculating our currency ratio
4. Constructing and executing our swap-and-add transaction

At the end of the guide, given the inputs above, we should be able swap-and-add liquidity using 100% of the input assets with the press of a button and see the change reflected in our position and the balance of our tokens.

For this guide, the following Uniswap packages are used:

- [@uniswap/v3-sdk](#)
- [@uniswap/sdk-core](#)
- [@uniswap/smart-order-router](#)

The core code of this guide can be found in [swapAndAddLiquidity\(\)](#).

:::note This guide assumes you are familiar with our [Minting a Position](#) guide. A minted position is required to add or remove liquidity from, so the buttons will be disabled until a position is minted.

Also note that we do not need to give approval to the `NonfungiblePositionManager` to transfer our tokens as we will have already done that when minting our position. :::

Setup a router instance

The first step is to approve the `SwapRouter` smart contract to spend our tokens for us in order for us to add liquidity to our position:

```
https://github.com/Uniswap/examples/blob/ec48bb845402419fa6e613cb26512a76d864afa5/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L58-L66
```

The we can setup our router, the `AlphaRouter`, which is part of the [smart-order-router package](#). The router requires a `chainId` and a `provider` to be initialized. Note that routing is not supported for local forks, so we will use a mainnet provider even when swapping on a local fork:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L57
```

For a more detailed example, check out our [routing guide](#).

Configuring our ratio calculation

Having created the router, we now need to construct the parameters required to make a call to its `routeToRatio` function, which will ensure the ratio of currency used matches the pool's required ratio to add our total liquidity. This will require the following parameters:

The first two parameters are the currency amounts we use as input to the `routeToRatio` algorithm:

```
https://github.com/Uniswap/examples/blob/c4667fadb13584268bbe2e0e0f556558a474751/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L78-L92
```

Next, we will create a placeholder position with a liquidity of `1` since liquidity is still unknown and will be set inside the call to `routeToRatio`:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L75-L78
```

We then need to create an instance of `SwapAndAddConfig` which will set additional configuration parameters for the `routeToRatio` algorithm:

- `ratioErrorTolerance` determines the margin of error the resulting ratio can have from the optimal ratio.
- `maxIterations` determines the maximum times the algorithm will iterate to find a ratio within error tolerance. If max iterations is exceeded, an error is returned. The benefit of running the algorithm more times is that we have more chances to find a route, but more iterations will longer to execute. We've used a default of 6 in our example.

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L80-L83
```

Finally, we will create an instance of `SwapAndAddOptions` to configure which position we are adding liquidity to and our defined swapping parameters in two different objects:

- `swapConfig` configures the `recipient` of leftover dust from swap, `slippageTolerance` and a `deadline` for the swap.
- `addLiquidityOptions` must contain a `tokenId` to add to an existing position

```
https://github.com/Uniswap/examples/blob/c4667fadb13584268bbe2e0e0f556558a474751/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L104-L114
```

Calculating our currency ratio

Having constructed all the parameters we need to call `routeToRatio`, we can now make the call to the function:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L97-L103
```

The return type of the function call is [SwapToRatioResponse](#). If a route was found successfully, this object will have two fields: the `status` (`success`) and the `SwapToRatioRoute` object. We check to make sure that both of those conditions hold true before we construct and submit the transaction:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L105-L110
```

In case a route was not found, we return from the function a `Failed` state for the transaction.

Constructing and executing our swap-and-add transaction

After making sure that a route was successfully found, we can now construct and send the transaction. The response (`SwapToRatioRoute`) will have the properties we need to construct our transaction object:

```
https://github.com/Uniswap/examples/blob/b5e64e3d6c17cb91bc081f1ed17581bbf22024bc/v3-sdk/swap-and-add-liquidity/src/libs/liquidity.ts#L112-L120
```

If the transaction was successful, our swap-and-add will be completed! We should see our input token balances decrease and our position balance should be increased accordingly.

id: overview sidebar_position: 1 title: Overview

The Uniswap V3 SDK

Welcome to the V3 Uniswap SDK!

The Uniswap V3 SDK provides abstractions to assist you with interacting with the Uniswap V3 smart contracts in a Typescript/Javascript environment (e.g. websites, node scripts). It makes uses of the [Core SDK](#) to gain access to abstractions that are common amongst the Uniswap SDKs. With the SDK, you can manipulate data that has been queried from the [EVM](#) using libraries that assist with needs such as data modeling, protection from rounding errors, and compile time enforced typing.

To begin, we recommend looking at our [Guides](#) which include [runnable examples](#) and walkthroughs of core usages. These guides will help you better understand how to use the SDK and integrate it into your application.

For complete documentation of the SDK's offerings, see the [Technical Reference](#).

Installation

To interact with the V3 SDK we recommend installing though npm:

```
npm i --save @uniswap/v3-sdk
npm i --save @uniswap/sdk-core
```

Developer Links

- [V3 SDK Github Repo](#)
- [Core SDK Github Repo](#)
- [V3 SDK NPM Package](#)

 Unit Tests passing  Lint passing npm@latest v3.9.0 minzipped size 113 kB discord join chat [@uniswap/v3-sdk / Exports / FullMath](#)

Class: FullMath

Table of contents

Constructors

- [constructor](#)

Methods

- [mulDivRoundingUp](#)

Constructors

constructor

- Private `new FullMath()`

Cannot be constructed.

Defined in

[utils/fullMath.ts:8](#)

Methods

mulDivRoundingUp

- Static `mulDivRoundingUp(a , b , denominator): default`

Parameters

| Name | Type |
|------|---------|
| a | default |
| b | default |

| | |
|-------------|---------|
| denominator | default |
|-------------|---------|

Returns

default

Defined in

[utils/fullMath.ts:10](#) @uniswap/v3-sdk / Exports / LiquidityMath

Class: LiquidityMath

Table of contents

Constructors

- [constructor](#)

Methods

- [addDelta](#)

Constructors

constructor

- `Private new LiquidityMath()`

Cannot be constructed.

Defined in

[utils/liquidityMath.ts:8](#)

Methods

addDelta

- `Static addDelta(x , y): default`

Parameters

| Name | Type |
|------|---------|
| x | default |
| y | default |

Returns

default

Defined in

[utils/liquidityMath.ts:10](#) @uniswap/v3-sdk / Exports / Multicall

Class: Multicall

Table of contents

Constructors

- [constructor](#)

Properties

- [INTERFACE](#)

Methods

- [encodeMulticall](#)

Constructors

constructor

- `Private new Multicall()`

Cannot be constructed.

Defined in

[multicall.ts:10](#)

Properties

INTERFACE

- `Static INTERFACE: Interface`

Defined in

[multicall.ts:5](#)

Methods

encodeMulticall

- `Static encodeMulticall(calldatas): string`

Parameters

| Name | Type |
|-----------|-------------------|
| calldatas | string string[] |

Returns

`string`

Defined in

[multicall.ts:12](#) @uniswap/v3-sdk / Exports / NoTickDataProvider

Class: NoTickDataProvider

This tick data provider does not know how to fetch any tick data. It throws whenever it is required. Useful if you do not need to load tick data for your use case.

Implements

- [TickDataProvider](#)

Table of contents

Constructors

- [constructor](#)

Properties

- [ERROR_MESSAGE](#)

Methods

- [getTick](#)
- [nextInitializedTickWithinOneWord](#)

Constructors

constructor

- `new NoTickDataProvider()`

Properties

ERROR_MESSAGE

- `Static Private ERROR_MESSAGE: string = 'No tick data provider was given'`

Defined in

[entities/tickDataProvider.ts:27](#)

Methods

getTick

• **getTick(_tick): Promise <{ liquidityNet : BigintIsh }>**

Return information corresponding to a specific tick

Parameters

| Name | Type | Description |
|-------|--------|------------------|
| _tick | number | the tick to load |

Returns

Promise <{ liquidityNet : BigintIsh }>

Implementation of

[TickDataProvider.getTick](#)

Defined in

[entities/tickDataProvider.ts:28](#)

nextInitializedTickWithinOneWord

• **nextInitializedTickWithinOneWord(_tick , _lte , _tickSpacing): Promise <[number , boolean]>**

Return the next tick that is initialized within a single word

Parameters

| Name | Type | Description |
|--------------|---------|--|
| _tick | number | The current tick |
| _lte | boolean | Whether the next tick should be lte the current tick |
| _tickSpacing | number | The tick spacing of the pool |

Returns

Promise <[number , boolean]>

Implementation of

[TickDataProvider.nextInitializedTickWithinOneWord](#)

Defined in

[entities/tickDataProvider.ts:32](#) @uniswap/v3-sdk / [Exports](#) / NonfungiblePositionManager

Class: NonfungiblePositionManager

Table of contents

Constructors

- [constructor](#)

Properties

- [INTERFACE](#)

Methods

- [addCallParameters](#)
- [collectCallParameters](#)
- [createCallParameters](#)
- [encodeCollect](#)
- [encodeCreate](#)
- [removeCallParameters](#)

- [safeTransferFromParameters](#)

Constructors

constructor

- `Private new NonfungiblePositionManager()`

Cannot be constructed.

Defined in

[nonfungiblePositionManager.ts:181](#)

Properties

INTERFACE

- `Static INTERFACE: Interface`

Defined in

[nonfungiblePositionManager.ts:176](#)

Methods

addCallParameters

- `Static addCallParameters(position , options): MethodParameters`

Parameters

| Name | Type |
|----------|-------------------------------------|
| position | Position |
| options | AddLiquidityOptions |

Returns

[MethodParameters](#)

Defined in

[nonfungiblePositionManager.ts:199](#)

collectCallParameters

- `Static collectCallParameters(options): MethodParameters`

Parameters

| Name | Type |
|---------|--------------------------------|
| options | CollectOptions |

Returns

[MethodParameters](#)

Defined in

[nonfungiblePositionManager.ts:326](#)

createCallParameters

- `Static createCallParameters(pool): MethodParameters`

Parameters

| Name | Type |
|------|----------------------|
| pool | Pool |

Returns

[MethodParameters](#)

Defined in

[nonfungiblePositionManager.ts:192](#)

encodeCollect

- Static Private **encodeCollect**(options): string []

Parameters

| Name | Type |
|---------|--------------------------------|
| options | CollectOptions |

Returns

string []

Defined in

[nonfungiblePositionManager.ts:286](#)

encodeCreate

- Static Private **encodeCreate**(pool): string

Parameters

| Name | Type |
|------|----------------------|
| pool | Pool |

Returns

string

Defined in

[nonfungiblePositionManager.ts:183](#)

removeCallParameters

- Static **removeCallParameters**(position , options): [MethodParameters](#)

Produces the calldata for completely or partially exiting a position

Parameters

| Name | Type | Description |
|----------|--|--|
| position | Position | The position to exit |
| options | RemoveLiquidityOptions | Additional information necessary for generating the calldata |

Returns

[MethodParameters](#)

The call parameters

Defined in

[nonfungiblePositionManager.ts:341](#)

safeTransferFromParameters

- Static **safeTransferFromParameters**(options): [MethodParameters](#)

Parameters

| Name | Type |
|---------|-------------------------------------|
| options | SafeTransferOptions |

Returns

[MethodParameters](#)

Defined in

[nonfungiblePositionManager.ts:416](#) @uniswap/v3-sdk / [Exports](#) / Payments

Class: Payments

Table of contents

Constructors

- [constructor](#)

Properties

- [INTERFACE](#)

Methods

- [encodeFeeBips](#)
- [encodeRefundETH](#)
- [encodeSweepToken](#)
- [encodeUnwrapWETH9](#)

Constructors

constructor

- Private `new Payments()`

Cannot be constructed.

Defined in

[payments.ts:25](#)

Properties

INTERFACE

- Static `INTERFACE: Interface`

Defined in

[payments.ts:20](#)

Methods

encodeFeeBips

- Static Private `encodeFeeBips(fee): string`

Parameters

| Name | Type |
|------|---------|
| fee | Percent |

Returns

`string`

Defined in

[payments.ts:27](#)

encodeRefundETH

- Static `encodeRefundETH(): string`

Returns

`string`

Defined in

[payments.ts:73](#)

encodeSweepToken

▸ Static **encodeSweepToken**(`token`, `amountMinimum`, `recipient`, `feeOptions?`): `string`

Parameters

| Name | Type |
|----------------------------|----------------------------|
| <code>token</code> | <code>Token</code> |
| <code>amountMinimum</code> | <code>default</code> |
| <code>recipient</code> | <code>string</code> |
| <code>feeOptions?</code> | FeeOptions |

Returns

`string`

Defined in

[payments.ts:49](#)

encodeUnwrapWETH9

▸ Static **encodeUnwrapWETH9**(`amountMinimum`, `recipient`, `feeOptions?`): `string`

Parameters

| Name | Type |
|----------------------------|----------------------------|
| <code>amountMinimum</code> | <code>default</code> |
| <code>recipient</code> | <code>string</code> |
| <code>feeOptions?</code> | FeeOptions |

Returns

`string`

Defined in

[payments.ts:31](#) @uniswap/v3-sdk / [Exports](#) / Pool

Class: Pool

Represents a V3 pool

Table of contents

Constructors

- [constructor](#)

Properties

- [_token0Price](#)
- [_token1Price](#)
- [fee](#)
- [liquidity](#)
- [sqrtRatioX96](#)
- [tickCurrent](#)
- [tickDataProvider](#)
- [token0](#)
- [token1](#)

Accessors

- [chainId](#)
- [tickSpacing](#)

- [token0Price](#)
- [token1Price](#)

Methods

- [getInputAmount](#)
- [getOutputAmount](#)
- [involvesToken](#)
- [priceOf](#)
- [swap](#)
- [getAddress](#)

Constructors

constructor

- **new Pool(tokenA , tokenB , fee , sqrtRatioX96 , liquidity , tickCurrent , ticks?)**

Construct a pool

Parameters

| Name | Type | Default value | Description |
|--------------|---|-------------------------------|---|
| tokenA | Token | undefined | One of the tokens in the pool |
| tokenB | Token | undefined | The other token in the pool |
| fee | FeeAmount | undefined | The fee in hundredths of a bips of the input amount of every swap that is collected by the pool |
| sqrtRatioX96 | Bigintish | undefined | The sqrt of the current ratio of amounts of token1 to token0 |
| liquidity | Bigintish | undefined | The current value of in range liquidity |
| tickCurrent | number | undefined | The current tick of the pool |
| ticks | TickDataProvider (Tick TickConstructorArgs)[] | NO_TICK_DATA_PROVIDER_DEFAULT | The current state of the pool ticks or a data provider that can return tick data |

Defined in

[entities/pool.ts:70](#)

Properties

_token0Price

- Private Optional **_token0Price**: Price < Token , Token >

Defined in

[entities/pool.ts:41](#)

_token1Price

- Private Optional **_token1Price**: Price < Token , Token >

Defined in

[entities/pool.ts:42](#)

fee

- Readonly **fee**: [FeeAmount](#)

Defined in

[entities/pool.ts:35](#)

liquidity

- Readonly **liquidity**: default

Defined in

[entities/pool.ts:37](#)

sqrtRatioX96

- `Readonly sqrtRatioX96: default`

Defined in

[entities/pool.ts:36](#)

tickCurrent

- `Readonly tickCurrent: number`

Defined in

[entities/pool.ts:38](#)

tickDataProvider

- `Readonly tickDataProvider: TickDataProvider`

Defined in

[entities/pool.ts:39](#)

token0

- `Readonly token0: Token`

Defined in

[entities/pool.ts:33](#)

token1

- `Readonly token1: Token`

Defined in

[entities/pool.ts:34](#)

Accessors

chainId

- `get chainId(): number`

Returns the chain ID of the tokens in the pool.

Returns

`number`

Defined in

[entities/pool.ts:149](#)

tickSpacing

- `get tickSpacing(): number`

Returns

`number`

Defined in

[entities/pool.ts:317](#)

token0Price

- `get token0Price(): Price < Token , Token >`

Returns the current mid price of the pool in terms of token0, i.e. the ratio of token1 over token0

Returns

```
Price < Token , Token >
```

Defined in

[entities/pool.ts:109](#)

token1Price

- `get token1Price(): Price < Token , Token >`

Returns the current mid price of the pool in terms of token1, i.e. the ratio of token0 over token1

Returns

```
Price < Token , Token >
```

Defined in

[entities/pool.ts:124](#)

Methods

getInputAmount

- `getInputAmount(outputAmount , sqrtPriceLimitX96?): Promise <[CurrencyAmount < Token >, Pool]>`

Given a desired output amount of a token, return the computed input amount and a pool with state updated after the trade

Parameters

| Name | Type | Description |
|--------------------|-----------------------|--|
| outputAmount | CurrencyAmount<Token> | the output amount for which to quote the input amount |
| sqrtPriceLimitX96? | default | The Q64.96 sqrt price limit. If zero for one, the price cannot be less than this value after the swap. If one for zero, the price cannot be greater than this value after the swap |

Returns

```
Promise <[ CurrencyAmount < Token >, Pool ]>
```

The input amount and the pool with updated state

Defined in

[entities/pool.ts:185](#)

getOutputAmount

- `getOutputAmount(inputAmount , sqrtPriceLimitX96?): Promise <[CurrencyAmount < Token >, Pool]>`

Given an input amount of a token, return the computed output amount, and a pool with state updated after the trade

Parameters

| Name | Type | Description |
|--------------------|-----------------------|---|
| inputAmount | CurrencyAmount<Token> | The input amount for which to quote the output amount |
| sqrtPriceLimitX96? | default | The Q64.96 sqrt price limit |

Returns

```
Promise <[ CurrencyAmount < Token >, Pool ]>
```

The output amount and the pool with updated state

Defined in

[entities/pool.ts:159](#)

involvesToken

- `involvesToken(token): boolean`

Returns true if the token is either token0 or token1

Parameters

| Name | Type | Description |
|-------|-------|--------------------|
| token | Token | The token to check |

Returns`boolean`

True if token is either token0 or token

Defined in[entities/pool.ts:102](#)**priceOf**

- `priceOf(token): Price < Token , Token >`

Return the price of the given token in terms of the other token in the pool.

Parameters

| Name | Type | Description |
|-------|-------|------------------------------|
| token | Token | The token to return price of |

Returns`Price < Token , Token >`

The price of the given token, in terms of the other.

Defined in[entities/pool.ts:141](#)**swap**

- `Private swap(zeroForOne , amountSpecified , sqrtPriceLimitX96?): Promise <{ amountCalculated : default ; liquidity : default ; sqrtRatioX96 : default ; tickCurrent : number }>`

Executes a swap

Parameters

| Name | Type | Description |
|--------------------|---------|--|
| zeroForOne | boolean | Whether the amount in is token0 or token1 |
| amountSpecified | default | The amount of the swap, which implicitly configures the swap as exact input (positive), or exact output (negative) |
| sqrtPriceLimitX96? | default | The Q64.96 sqrt price limit. If zero for one, the price cannot be less than this value after the swap. If one for zero, the price cannot be greater than this value after the swap |

Returns`Promise <{ amountCalculated : default ; liquidity : default ; sqrtRatioX96 : default ; tickCurrent : number }>`

amountCalculated

sqrtRatioX96

liquidity

tickCurrent

Defined in[entities/pool.ts:215](#)**getAddress**

- `Static getAddress(tokenA , tokenB , fee , initCodeHashManualOverride? , factoryAddressOverride?): string`

Parameters

| Name | Type |
|-----------------------------|---------------------------|
| tokenA | Token |
| tokenB | Token |
| fee | FeeAmount |
| initCodeHashManualOverride? | string |
| factoryAddressOverride? | string |

Returns`string`**Defined in**[entities/pool.ts:44 @uniswap/v3-sdk / Exports / Position](#)

Class: Position

Represents a position on a Uniswap V3 Pool

Table of contents

Constructors

- [constructor](#)

Properties

- [_mintAmounts](#)
- [_token0Amount](#)
- [_token1Amount](#)
- [liquidity](#)
- [pool](#)
- [tickLower](#)
- [tickUpper](#)

Accessors

- [amount0](#)
- [amount1](#)
- [mintAmounts](#)
- [token0PriceLower](#)
- [token0PriceUpper](#)

Methods

- [burnAmountsWithSlippage](#)
- [mintAmountsWithSlippage](#)
- [ratiosAfterSlippage](#)
- [fromAmount0](#)
- [fromAmount1](#)
- [fromAmounts](#)

Constructors

constructor

- **new Position(`__namedParameters`)**

Constructs a position for a given pool with the given liquidity

Parameters

| Name | Type |
|--------------------------------|--------------------------------------|
| <code>__namedParameters</code> | <code>PositionConstructorArgs</code> |

Defined in[entities/position.ts:40](#)

Properties

_mintAmounts

- Private `_mintAmounts: null | Readonly<{ amount0: default ; amount1: default }> = null`

Defined in

[entities/position.ts:31](#)

_token0Amount

- Private `_token0Amount: null | CurrencyAmount < Token > = null`

Defined in

[entities/position.ts:29](#)

_token1Amount

- Private `_token1Amount: null | CurrencyAmount < Token > = null`

Defined in

[entities/position.ts:30](#)

liquidity

- Readonly `liquidity: default`

Defined in

[entities/position.ts:26](#)

pool

- Readonly `pool: Pool`

Defined in

[entities/position.ts:23](#)

tickLower

- Readonly `tickLower: number`

Defined in

[entities/position.ts:24](#)

tickUpper

- Readonly `tickUpper: number`

Defined in

[entities/position.ts:25](#)

Accessors

amount0

- get `amount0(): CurrencyAmount < Token >`

Returns the amount of token0 that this position's liquidity could be burned for at the current pool price

Returns

`CurrencyAmount < Token >`

Defined in

[entities/position.ts:68](#)

amount1

- get **amount1()**: `CurrencyAmount < Token >`

Returns the amount of token1 that this position's liquidity could be burned for at the current pool price

Returns

`CurrencyAmount < Token >`

Defined in

[entities/position.ts:100](#)

mintAmounts

- get **mintAmounts()**: `Readonly <{ amount0 : default ; amount1 : default }>`

Returns the minimum amounts that must be sent in order to mint the amount of liquidity held by the position at the current price for the pool

Returns

`Readonly <{ amount0 : default ; amount1 : default }>`

Defined in

[entities/position.ts:258](#)

token0PriceLower

- get **token0PriceLower()**: `Price < Token , Token >`

Returns the price of token0 at the lower tick

Returns

`Price < Token , Token >`

Defined in

[entities/position.ts:54](#)

token0PriceUpper

- get **token0PriceUpper()**: `Price < Token , Token >`

Returns the price of token0 at the upper tick

Returns

`Price < Token , Token >`

Defined in

[entities/position.ts:61](#)

Methods

burnAmountsWithSlippage

- **burnAmountsWithSlippage(slippageTolerance)**: `Readonly <{ amount0 : default ; amount1 : default }>`

Returns the minimum amounts that should be requested in order to safely burn the amount of liquidity held by the position with the given slippage tolerance

Parameters

| Name | Type | Description |
|-------------------|---------|--|
| slippageTolerance | Percent | tolerance of unfavorable slippage from the current price |

Returns

`Readonly <{ amount0 : default ; amount1 : default }>`

The amounts, with slippage

Defined in

[entities/position.ts:213](#)

mintAmountsWithSlippage

▪ **mintAmountsWithSlippage(slippageTolerance): Readonly <{ amount0 : default ; amount1 : default }>**

Returns the minimum amounts that must be sent in order to safely mint the amount of liquidity held by the position with the given slippage tolerance

Parameters

| Name | Type | Description |
|-------------------|---------|--|
| slippageTolerance | Percent | Tolerance of unfavorable slippage from the current price |

Returns

Readonly <{ amount0 : default ; amount1 : default }>

The amounts, with slippage

Defined in

[entities/position.ts:157](#)

ratiosAfterSlippage

▪ **Private ratiosAfterSlippage(slippageTolerance): Object**

Returns the lower and upper sqrt ratios if the price 'slips' up to slippage tolerance percentage

Parameters

| Name | Type | Description |
|-------------------|---------|---|
| slippageTolerance | Percent | The amount by which the price can 'slip' before the transaction will revert |

Returns

Object

The sqrt ratios after slippage

| Name | Type |
|-------------------|---------|
| sqrtRatioX96Lower | default |
| sqrtRatioX96Upper | default |

Defined in

[entities/position.ts:134](#)

fromAmount0

▪ **Static fromAmount0(__namedParameters): [Position](#)**

Computes a position with the maximum amount of liquidity received for a given amount of token0, assuming an unlimited amount of token1

Parameters

| Name | Type |
|------------------------------------|----------------------|
| __namedParameters | Object |
| __namedParameters.amount0 | BigintIsh |
| __namedParameters.pool | Pool |
| __namedParameters.tickLower | number |
| __namedParameters.tickUpper | number |
| __namedParameters.useFullPrecision | boolean |

Returns

[Position](#)

The position

Defined in

[entities/position.ts:354](#)

fromAmount1

▸ Static **fromAmount1**(namedParameters): [Position](#)

Computes a position with the maximum amount of liquidity received for a given amount of token1, assuming an unlimited amount of token0

Parameters

| Name | Type |
|----------------------------------|----------------------|
| <u>namedParameters</u> | Object |
| <u>namedParameters.amount1</u> | BigintIsh |
| <u>namedParameters.pool</u> | Pool |
| <u>namedParameters.tickLower</u> | number |
| <u>namedParameters.tickUpper</u> | number |

Returns

[Position](#)

The position

Defined in

[entities/position.ts:378](#)

fromAmounts

▸ Static **fromAmounts**(namedParameters): [Position](#)

Computes the maximum amount of liquidity received for a given amount of token0, token1, and the prices at the tick boundaries.

Parameters

| Name | Type |
|---|----------------------|
| <u>namedParameters</u> | Object |
| <u>namedParameters.amount0</u> | BigintIsh |
| <u>namedParameters.amount1</u> | BigintIsh |
| <u>namedParameters.pool</u> | Pool |
| <u>namedParameters.tickLower</u> | number |
| <u>namedParameters.tickUpper</u> | number |
| <u>namedParameters.useFullPrecision</u> | boolean |

Returns

[Position](#)

The amount of liquidity for the position

Defined in

[entities/position.ts:312](#) [@uniswap/v3-sdk](#) / [Exports](#) / PositionLibrary

Class: PositionLibrary

Table of contents

Constructors

- [constructor](#)

Methods

- [getTokensOwed](#)

Constructors

constructor

- `Private new PositionLibrary()`

Cannot be constructed.

Defined in

[utils/position.ts:10](#)

Methods

getTokensOwed

- `Static getTokensOwed(feeGrowthInside0LastX128 , feeGrowthInside1LastX128 , liquidity , feeGrowthInside0X128 , feeGrowthInside1X128): default []`

Parameters

| Name | Type |
|---------------------------------------|----------------------|
| <code>feeGrowthInside0LastX128</code> | <code>default</code> |
| <code>feeGrowthInside1LastX128</code> | <code>default</code> |
| <code>liquidity</code> | <code>default</code> |
| <code>feeGrowthInside0X128</code> | <code>default</code> |
| <code>feeGrowthInside1X128</code> | <code>default</code> |

Returns

- `default []`

Defined in

[utils/position.ts:13](#) `@uniswap/v3-sdk / Exports / Route`

Class: Route<TInput, TOutput>

Represents a list of pools through which a swap can occur

Type parameters

| Name | Type | Description |
|----------------------|-------------------------------|------------------|
| <code>TInput</code> | <code>extends Currency</code> | The input token |
| <code>TOutput</code> | <code>extends Currency</code> | The output token |

Table of contents

Constructors

- [constructor](#)

Properties

- [_midPrice](#)
- [input](#)
- [output](#)
- [pools](#)
- [tokenPath](#)

Accessors

- [chainId](#)

- [midPrice](#)

Constructors

constructor

- `new Route< TInput , TOutput >(pools , input , output)`

Creates an instance of route.

Type parameters

| Name | Type |
|---------|------------------|
| TInput | extends Currency |
| TOutput | extends Currency |

Parameters

| Name | Type | Description |
|--------|------------------------|--|
| pools | Pool[] | An array of <code>Pool</code> objects, ordered by the route the swap will take |
| input | TInput | The input token |
| output | TOutput | The output token |

Defined in

[entities/route.ts:25](#)

Properties

_midPrice

- Private `_midPrice: null | Price < TInput , TOutput > = null`

Defined in

[entities/route.ts:17](#)

input

- Readonly `input: TInput`

Defined in

[entities/route.ts:14](#)

output

- Readonly `output: TOutput`

Defined in

[entities/route.ts:15](#)

pools

- Readonly `pools: Pool[]`

Defined in

[entities/route.ts:12](#)

tokenPath

- Readonly `tokenPath: Token[]`

Defined in

[entities/route.ts:13](#)

Accessors

chainId

- `get chainId(): number`

Returns

`number`

Defined in

[entities/route.ts:54](#)

midPrice

- `get midPrice(): Price < TInput , TOutput >`

Returns the mid price of the route

Returns

`Price < TInput , TOutput >`

Defined in

[entities/route.ts:61](#) [@uniswap/v3-sdk](#) / [Exports](#) / SelfPermit

Class: SelfPermit

Table of contents

Constructors

- [constructor](#)

Properties

- [INTERFACE](#)

Methods

- [encodePermit](#)

Constructors

constructor

- `Private new SelfPermit()`

Cannot be constructed.

Defined in

[selfPermit.ts:34](#)

Properties

INTERFACE

- `Static INTERFACE: Interface`

Defined in

[selfPermit.ts:29](#)

Methods

encodePermit

- `Static encodePermit(token , options): string`

Parameters

| Name | Type |
|---------|-------------------------------|
| token | Token |
| options | PermitOptions |

Returns`string`**Defined in**[selfPermit.ts:36 @uniswap/v3-sdk / Exports / SqrtPriceMath](#)

Class: SqrtPriceMath

Table of contents

Constructors

- [constructor](#)

Methods

- [getAmount0Delta](#)
- [getAmount1Delta](#)
- [getNextSqrtPriceFromAmount0RoundingUp](#)
- [getNextSqrtPriceFromAmount1RoundingDown](#)
- [getNextSqrtPriceFromInput](#)
- [getNextSqrtPriceFromOutput](#)

Constructors

constructor

- `Private new SqrtPriceMath()`

Cannot be constructed.

Defined in[utils/sqrtPriceMath.ts:23](#)

Methods

getAmount0Delta

- `Static getAmount0Delta(sqrtRatioAX96 , sqrtRatioBX96 , liquidity , roundUp): default`

Parameters

| Name | Type |
|---------------|---------|
| sqrtRatioAX96 | default |
| sqrtRatioBX96 | default |
| liquidity | default |
| roundUp | boolean |

Returns`default`**Defined in**[utils/sqrtPriceMath.ts:25](#)

getAmount1Delta

- `Static getAmount1Delta(sqrtRatioAX96 , sqrtRatioBX96 , liquidity , roundUp): default`

Parameters

| Name | Type |
|---------------|---------|
| sqrtRatioAX96 | default |
| sqrtRatioBX96 | default |
| liquidity | default |

| | |
|---------|---------|
| roundUp | boolean |
|---------|---------|

Returns

default

Defined in

[utils/sqrtPriceMath.ts:38](#)

getNextSqrtPriceFromAmount0RoundingUp

▸ Static Private **getNextSqrtPriceFromAmount0RoundingUp**(sqrtPX96 , liquidity , amount , add): default

Parameters

| Name | Type |
|-----------|---------|
| sqrtPX96 | default |
| liquidity | default |
| amount | default |
| add | boolean |

Returns

default

Defined in

[utils/sqrtPriceMath.ts:71](#)

getNextSqrtPriceFromAmount1RoundingDown

▸ Static Private **getNextSqrtPriceFromAmount1RoundingDown**(sqrtPX96 , liquidity , amount , add): default

Parameters

| Name | Type |
|-----------|---------|
| sqrtPX96 | default |
| liquidity | default |
| amount | default |
| add | boolean |

Returns

default

Defined in

[utils/sqrtPriceMath.ts:100](#)

getNextSqrtPriceFromInput

▸ Static **getNextSqrtPriceFromInput**(sqrtPX96 , liquidity , amountIn , zeroForOne): default

Parameters

| Name | Type |
|------------|---------|
| sqrtPX96 | default |
| liquidity | default |
| amountIn | default |
| zeroForOne | boolean |

Returns

```
default
```

Defined in

[utils/sqrtPriceMath.ts:48](#)

getNextSqrtPriceFromOutput

- Static `getNextSqrtPriceFromOutput(sqrtPX96 , liquidity , amountOut , zeroForOne): default`

Parameters

| Name | Type |
|------------|---------|
| sqrtPX96 | default |
| liquidity | default |
| amountOut | default |
| zeroForOne | boolean |

Returns

```
default
```

Defined in

[utils/sqrtPriceMath.ts:57](#) @uniswap/v3-sdk / Exports / Staker

Class: Staker

Table of contents

Constructors

- [constructor](#)

Properties

- [INCENTIVE_KEY_ABI](#)
- [INTERFACE](#)

Methods

- [_encodeIncentiveKey](#)
- [collectRewards](#)
- [encodeClaim](#)
- [encodeDeposit](#)
- [withdrawToken](#)

Constructors

constructor

- Protected `new Staker()`

Defined in

[staker.ts:72](#)

Properties

INCENTIVE_KEY_ABI

- Static Private `INCENTIVE_KEY_ABI: string = 'tuple(address rewardToken, address pool, uint256 startTime, uint256 endTime, address refundee)'`

Defined in

[staker.ts:73](#)

INTERFACE

- Static `INTERFACE: Interface`

Defined in

[staker.ts:70](#)

Methods

_encodeIncentiveKey

▪ Static Private `_encodeIncentiveKey(incentiveKey): Object`

Parameters

| Name | Type | Description |
|--------------|------------------------------|--|
| incentiveKey | IncentiveKey | An IncentiveKey which represents a unique staking program. |

Returns

`Object`

An encoded IncentiveKey to be read by ethers

Defined in

[staker.ts:194](#)

collectRewards

▪ Static `collectRewards(incentiveKeys , options): MethodParameters`

Note: A `tokenId` can be staked in many programs but to claim rewards and continue the program you must unstake, claim, and then restate.

Parameters

| Name | Type | Description |
|---------------|---|--|
| incentiveKeys | IncentiveKey IncentiveKey[] | An IncentiveKey or array of IncentiveKeys that <code>tokenId</code> is staked in. Input an array of IncentiveKeys to claim rewards for each program. |
| options | ClaimOptions | ClaimOptions to specify <code>tokenId</code> , recipient, and amount wanting to collect. Note that you can only specify one amount and one recipient across the various programs if you are collecting from multiple programs at once. |

Returns

[MethodParameters](#)

Defined in

[staker.ts:107](#)

encodeClaim

▪ Static Private `encodeClaim(incentiveKey , options): string []`

To claim rewards, must unstake and then claim.

Parameters

| Name | Type | Description |
|--------------|------------------------------|--|
| incentiveKey | IncentiveKey | The unique identifier of a staking program. |
| options | ClaimOptions | Options for producing the calldata to claim. Can't claim unless you unstake. |

Returns

`string []`

The calldatas for 'unstakeToken' and 'claimReward'.

Defined in

[staker.ts:82](#)

encodeDeposit

▸ Static **encodeDeposit**(incentiveKeys): string

Parameters

| Name | Type | Description |
|---------------|---|---|
| incentiveKeys | IncentiveKey IncentiveKey[] | A single IncentiveKey or array of IncentiveKeys to be encoded and used in the data parameter in <code>safeTransferFrom</code> |

Returns

string

An IncentiveKey as a string

Defined in

[staker.ts:173](#)

withdrawToken

▸ Static **withdrawToken**(incentiveKeys , withdrawOptions): [MethodParameters](#)

Parameters

| Name | Type | Description |
|-----------------|---|--|
| incentiveKeys | IncentiveKey IncentiveKey[] | A list of incentiveKeys to unstake from. Should include all incentiveKeys (unique staking programs) that options.tokenId is staked in. |
| withdrawOptions | FullWithdrawOptions | Options for producing claim calldata and withdraw calldata. Can't withdraw without unstaking all programs for <code>tokenId</code> . |

Returns

[MethodParameters](#)

Calldata for unstaking, claiming, and withdrawing.

Defined in

[staker.ts:136](#) @uniswap/v3-sdk / [Exports](#) / SwapMath

Class: SwapMath

Table of contents

Constructors

- [constructor](#)

Methods

- [computeSwapStep](#)

Constructors

constructor

• Private `new SwapMath()`

Cannot be constructed.

Defined in

[utils/swapMath.ts:13](#)

Methods

computeSwapStep

▸ Static **computeSwapStep**(sqrtRatioCurrentX96 , sqrtRatioTargetX96 , liquidity , amountRemaining , feePips): [default , default , default , default]

Parameters

| Name | Type |
|---------------------|---------------------------|
| sqrtRatioCurrentX96 | default |
| sqrtRatioTargetX96 | default |
| liquidity | default |
| amountRemaining | default |
| feePips | FeeAmount |

Returns

[default , default , default , default]

Defined in

[utils/swapMath.ts:15](#) @uniswap/v3-sdk / [Exports](#) / SwapQuoter

Class: SwapQuoter

Represents the Uniswap V3 QuoterV1 contract with a method for returning the formatted calldata needed to call the quoter contract.

Table of contents

Constructors

- [constructor](#)

Properties

- [V1INTERFACE](#)
- [V2INTERFACE](#)

Methods

- [quoteCallParameters](#)

Constructors

constructor

- new [SwapQuoter\(\)](#)

Properties

V1INTERFACE

- Static [V1INTERFACE](#): Interface

Defined in

[quoter.ts:37](#)

V2INTERFACE

- Static [V2INTERFACE](#): Interface

Defined in

[quoter.ts:38](#)

Methods

quoteCallParameters

- Static [quoteCallParameters](#)< TInput , TOutput >(route , amount , tradeType , options?): [MethodParameters](#)

Produces the on-chain method name of the appropriate function within QuoterV2, and the relevant hex encoded parameters.

Type parameters

| Name | Type | Description |
|--------|----------------------------------|--|
| TInput | extends Currency | The input token, either Ether or an ERC-20 |

| | | |
|---------|------------------|---|
| TOutput | extends Currency | The output token, either Ether or an ERC-20 |
|---------|------------------|---|

Parameters

| Name | Type | Description |
|-----------|--|--|
| route | Route<TInput, TOutput> | The swap route, a list of pools through which a swap can occur |
| amount | CurrencyAmount<TInput TOutput> | The amount of the quote, either an amount in, or an amount out |
| tradeType | TradeType | The trade type, either exact input or exact output |
| options | QuoteOptions | The optional params including price limit and Quoter contract switch |

Returns

[MethodParameters](#)

The formatted calldata

Defined in

[quoter.ts:51 @uniswap/v3-sdk / Exports](#) / SwapRouter

Class: SwapRouter

Represents the Uniswap V3 SwapRouter, and has static methods for helping execute trades.

Table of contents

Constructors

- [constructor](#)

Properties

- [INTERFACE](#)

Methods

- [swapCallParameters](#)

Constructors

constructor

- Private `new SwapRouter()`

Cannot be constructed.

Defined in

[swapRouter.ts:57](#)

Properties

INTERFACE

- Static `INTERFACE: Interface`

Defined in

[swapRouter.ts:52](#)

Methods

swapCallParameters

- Static `swapCallParameters(trades , options): MethodParameters`

Produces the on-chain method name to call and the hex encoded parameters to pass as arguments for a given trade.

Parameters

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

| | | |
|---------|---|---------------------------------|
| trades | Trade <Currency, Currency, TradeType> Trade <Currency, Currency, TradeType>[] | - |
| options | SwapOptions | options for the call parameters |

Returns

[MethodParameters](#)

Defined in

[swapRouter.ts:64 @uniswap/v3-sdk / Exports](#) / Tick

Class: Tick

Table of contents

Constructors

- [constructor](#)

Properties

- [index](#)
- [liquidityGross](#)
- [liquidityNet](#)

Constructors

constructor

- `new Tick(__namedParameters)`

Parameters

| Name | Type |
|--------------------------------|-------------------------------------|
| <code>__namedParameters</code> | TickConstructorArgs |

Defined in

[entities/tick.ts:17](#)

Properties

index

- `Readonly index: number`

Defined in

[entities/tick.ts:13](#)

liquidityGross

- `Readonly liquidityGross: default`

Defined in

[entities/tick.ts:14](#)

liquidityNet

- `Readonly liquidityNet: default`

Defined in

[entities/tick.ts:15 @uniswap/v3-sdk / Exports](#) / TickLibrary

Class: TickLibrary

Table of contents

Constructors

- [constructor](#)

Methods

- [getFeeGrowthInside](#)

Constructors

constructor

- Private `new TickLibrary()`

Cannot be constructed.

Defined in

[utils/tickLibrary.ts:25](#)

Methods

getFeeGrowthInside

- Static `getFeeGrowthInside(feeGrowthOutsideLower , feeGrowthOutsideUpper , tickLower , tickUpper , tickCurrent , feeGrowthGlobal0X128 , feeGrowthGlobal1X128): default []`

Parameters

| Name | Type |
|------------------------------------|-------------------------------|
| <code>feeGrowthOutsideLower</code> | <code>FeeGrowthOutside</code> |
| <code>feeGrowthOutsideUpper</code> | <code>FeeGrowthOutside</code> |
| <code>tickLower</code> | <code>number</code> |
| <code>tickUpper</code> | <code>number</code> |
| <code>tickCurrent</code> | <code>number</code> |
| <code>feeGrowthGlobal0X128</code> | <code>default</code> |
| <code>feeGrowthGlobal1X128</code> | <code>default</code> |

Returns

`default []`

Defined in

[utils/tickLibrary.ts:27](#) [@uniswap/v3-sdk / Exports](#) / `TickList`

Class: TickList

Utility methods for interacting with sorted lists of ticks

Table of contents

Constructors

- [constructor](#)

Methods

- [binarySearch](#)
- [getTick](#)
- [isAtOrAboveLargest](#)
- [isBelowSmallest](#)
- [nextInitializedTick](#)
- [nextInitializedTickWithinOneWord](#)
- [validateList](#)

Constructors

constructor

- Private `new TickList()`

Cannot be constructed

Defined in

[utils/tickList.ts:18](#)

Methods

binarySearch

- Static Private **binarySearch**(ticks , tick): number

Finds the largest tick in the list of ticks that is less than or equal to tick

Parameters

| Name | Type | Description |
|-------|---------------------------------|--|
| ticks | readonly tick[] | list of ticks |
| tick | number | tick to find the largest tick that is less than or equal to tick |

Returns

number

Defined in

[utils/tickList.ts:62](#)

getTick

- Static **getTick**(ticks , index): [Tick](#)

Parameters

| Name | Type |
|-------|---------------------------------|
| ticks | readonly tick[] |
| index | number |

Returns

[Tick](#)

Defined in

[utils/tickList.ts:50](#)

isAtOrAboveLargest

- Static **isAtOrAboveLargest**(ticks , tick): boolean

Parameters

| Name | Type |
|-------|---------------------------------|
| ticks | readonly tick[] |
| tick | number |

Returns

boolean

Defined in

[utils/tickList.ts:45](#)

isBelowSmallest

- Static **isBelowSmallest**(ticks , tick): boolean

Parameters

| Name | Type |
|------|------|
| | |

| | |
|-------|---------------------------------|
| ticks | readonly Tick[] |
| tick | number |

Returns

`boolean`

Defined in

[utils/tickList.ts:40](#)

nextInitializedTick

- Static `nextInitializedTick(ticks , tick , lte): Tick`

Parameters

| Name | Type |
|-------|---------------------------------|
| ticks | readonly Tick[] |
| tick | number |
| lte | boolean |

Returns

`Tick`

Defined in

[utils/tickList.ts:83](#)

nextInitializedTickWithinOneWord

- Static `nextInitializedTickWithinOneWord(ticks , tick , lte , tickSpacing): [number , boolean]`

Parameters

| Name | Type |
|-------------|---------------------------------|
| ticks | readonly Tick[] |
| tick | number |
| lte | boolean |
| tickSpacing | number |

Returns

`[number , boolean]`

Defined in

[utils/tickList.ts:101](#)

validateList

- Static `validateList(ticks , tickSpacing): void`

Parameters

| Name | Type |
|-------------|------------------------|
| ticks | Tick[] |
| tickSpacing | number |

Returns

`void`

Defined in

[utils/tickList.ts:20](#) @uniswap/v3-sdk / [Exports](#) / TickListDataProvider

Class: TickListDataProvider

A data provider for ticks that is backed by an in-memory array of ticks.

Implements

- [TickDataProvider](#)

Table of contents

Constructors

- [constructor](#)

Properties

- [ticks](#)

Methods

- [getTick](#)
- [nextInitializedTickWithinOneWord](#)

Constructors

constructor

- `new TickListDataProvider(ticks , tickSpacing)`

Parameters

| Name | Type |
|-------------|--|
| ticks | (Tick TickConstructorArgs)[] |
| tickSpacing | number |

Defined in

[entities/tickListDataProvider.ts:12](#)

Properties

ticks

- `Private ticks: readonly Tick []`

Defined in

[entities/tickListDataProvider.ts:10](#)

Methods

getTick

- `getTick(tick): Promise <{ liquidityGross : BigintIsh ; liquidityNet : BigintIsh }>`

Return information corresponding to a specific tick

Parameters

| Name | Type | Description |
|------|--------|------------------|
| tick | number | the tick to load |

Returns

`Promise <{ liquidityGross : BigintIsh ; liquidityNet : BigintIsh }>`

Implementation of

[TickDataProvider.getTick](#)

Defined in

[entities/tickListDataProvider.ts:18](#)

nextInitializedTickWithinOneWord

▸ **nextInitializedTickWithinOneWord**(`tick` : number, `lte` : boolean, `tickSpacing` : number) : Promise<[number, boolean]>

Return the next tick that is initialized within a single word

Parameters

| Name | Type | Description |
|--------------------------|---------|--|
| <code>tick</code> | number | The current tick |
| <code>lte</code> | boolean | Whether the next tick should be lte the current tick |
| <code>tickSpacing</code> | number | The tick spacing of the pool |

Returns

Promise<[number, boolean]>

Implementation of

[TickDataProvider.nextInitializedTickWithinOneWord](#)

Defined in

[entities/tickListDataProvider.ts:22](#) @uniswap/v3-sdk / [Exports](#) / TickMath

Class: TickMath

Table of contents

Constructors

- [constructor](#)

Properties

- [MAX_SQRT_RATIO](#)
- [MAX_TICK](#)
- [MIN_SQRT_RATIO](#)
- [MIN_TICK](#)

Methods

- [getSqrtRatioAtTick](#)
- [getTickAtSqrtRatio](#)

Constructors

constructor

• `Private new TickMath()`

Cannot be constructed.

Defined in

[utils/tickMath.ts:17](#)

Properties

MAX_SQRT_RATIO

• `Static MAX_SQRT_RATIO: default`

The sqrt ratio corresponding to the maximum tick that could be used on any pool.

Defined in

[utils/tickMath.ts:35](#)

MAX_TICK

- Static **MAX_TICK**: number = -TickMath.MIN_TICK

The maximum tick that can be used on any pool.

Defined in

[utils/tickMath.ts:26](#)

MIN_SQRT_RATIO

- Static **MIN_SQRT_RATIO**: default

The sqrt ratio corresponding to the minimum tick that could be used on any pool.

Defined in

[utils/tickMath.ts:31](#)

MIN_TICK

- Static **MIN_TICK**: number = -887272

The minimum tick that can be used on any pool.

Defined in

[utils/tickMath.ts:22](#)

Methods

getSqrtRatioAtTick

- Static **getSqrtRatioAtTick(tick)**: default

Returns the sqrt ratio as a Q64.96 for the given tick. The sqrt ratio is computed as $\sqrt{1.0001}^{\text{tick}}$

Parameters

| Name | Type | Description |
|------|--------|--|
| tick | number | the tick for which to compute the sqrt ratio |

Returns

default

Defined in

[utils/tickMath.ts:41](#)

getTickAtSqrtRatio

- Static **getTickAtSqrtRatio(sqrtRatioX96)**: number

Returns the tick corresponding to a given sqrt ratio, s.t. $\#getSqrtRatioAtTick(\text{tick}) \leq \text{sqrtRatioX96}$ and $\#getSqrtRatioAtTick(\text{tick} + 1) > \text{sqrtRatioX96}$

Parameters

| Name | Type | Description |
|--------------|---------|--|
| sqrtRatioX96 | default | the sqrt ratio as a Q64.96 for which to compute the tick |

Returns

number

Defined in

[utils/tickMath.ts:82](#) @uniswap/v3-sdk / [Exports](#) / Trade

Class: Trade<TInput, TOutput, TTradeType>

Represents a trade executed against a set of routes where some percentage of the input is split across each route.

Each route has its own set of pools. Pools can not be re-used across routes.

Does not account for slippage, i.e., changes in price environment that can occur between the time the trade is submitted and when it is executed.

Type parameters

| Name | Type | Description |
|------------|-------------------|--|
| TInput | extends Currency | The input token, either Ether or an ERC-20 |
| TOutput | extends Currency | The output token, either Ether or an ERC-20 |
| TTradeType | extends TradeType | The trade type, either exact input or exact output |

Table of contents

Constructors

- [constructor](#)

Properties

- [executionPrice](#)
- [inputAmount](#)
- [outputAmount](#)
- [priceImpact](#)
- [swaps](#)
- [tradeType](#)

Accessors

- [executionPrice](#)
- [inputAmount](#)
- [outputAmount](#)
- [priceImpact](#)
- [route](#)

Methods

- [maximumAmountIn](#)
- [minimumAmountOut](#)
- [worstExecutionPrice](#)
- [bestTradeExactIn](#)
- [bestTradeExactOut](#)
- [createUncheckedTrade](#)
- [createUncheckedTradeWithMultipleRoutes](#)
- [exactIn](#)
- [exactOut](#)
- [fromRoute](#)
- [fromRoutes](#)

Constructors

constructor

```
• Private new Trade< TInput , TOutput , TTradeType >( __namedParameters )
```

Construct a trade by passing in the pre-computed property values

Type parameters

| Name | Type |
|------------|-------------------|
| TInput | extends Currency |
| TOutput | extends Currency |
| TTradeType | extends TradeType |

Parameters

| Name | Type |
|--------------------------|--|
| __namedParameters | Object |
| __namedParameters.routes | { inputAmount: CurrencyAmount<TInput> ; outputAmount: CurrencyAmount<TOutput> ; route: Route <TInput, TOutput> }[] |

| | |
|--|-------------------------|
| <code>__namedParameters.tradeType</code> | <code>TTradeType</code> |
|--|-------------------------|

Defined in

[entities/trade.ts:397](#)

Properties

`_executionPrice`

- `Private _executionPrice: undefined | Price < TInput , TOutput >`

The cached result of the computed execution price

Defined in

[entities/trade.ts:143](#)

`_inputAmount`

- `Private _inputAmount: undefined | CurrencyAmount < TInput >`

The cached result of the input amount computation

Defined in

[entities/trade.ts:97](#)

`_outputAmount`

- `Private _outputAmount: undefined | CurrencyAmount < TOutput >`

The cached result of the output amount computation

Defined in

[entities/trade.ts:120](#)

`_priceImpact`

- `Private _priceImpact: undefined | Percent`

The cached result of the price impact computation

Defined in

[entities/trade.ts:164](#)

`swaps`

- `Readonly swaps: { inputAmount : CurrencyAmount < TInput >; outputAmount : CurrencyAmount < TOutput >; route : Route < TInput , TOutput > }[]`

The swaps of the trade, i.e. which routes and how much is swapped in each that make up the trade.

Defined in

[entities/trade.ts:82](#)

`tradeType`

- `Readonly tradeType: TTradeType`

The type of the trade, either exact in or exact out.

Defined in

[entities/trade.ts:91](#)

Accessors

`executionPrice`

- `get executionPrice(): Price < TInput , TOutput >`

The price expressed in terms of output amount/input amount.

Returns

```
Price < TInput , TOutput >
```

Defined in

[entities/trade.ts:148](#)

inputAmount

- `get inputAmount(): CurrencyAmount < TInput >`

The input amount for the trade assuming no slippage.

Returns

```
CurrencyAmount < TInput >
```

Defined in

[entities/trade.ts:102](#)

outputAmount

- `get outputAmount(): CurrencyAmount < TOutput >`

The output amount for the trade assuming no slippage.

Returns

```
CurrencyAmount < TOutput >
```

Defined in

[entities/trade.ts:125](#)

priceImpact

- `get priceImpact(): Percent`

Returns the percent difference between the route's mid price and the price impact

Returns

```
Percent
```

Defined in

[entities/trade.ts:169](#)

route

- `get route(): Route < TInput , TOutput >`

Deprecated

Deprecated in favor of 'swaps' property. If the trade consists of multiple routes this will return an error.

When the trade consists of just a single route, this returns the route of the trade, i.e. which pools the trade goes through.

Returns

```
Route < TInput , TOutput >
```

Defined in

[entities/trade.ts:73](#)

Methods

maximumAmountIn

- `maximumAmountIn(slippageTolerance , amountIn?): CurrencyAmount < TInput >`

Get the maximum amount in that can be spent via this trade for the given slippage tolerance

Parameters

| Name | Type | Description |
|-------------------|------------------------|--|
| slippageTolerance | Percent | The tolerance of unfavorable slippage from the execution price of this trade |
| amountIn | CurrencyAmount<TInput> | - |

Returns

`CurrencyAmount < TInput >`

The amount in

Defined in

[entities/trade.ts:456](#)

minimumAmountOut

▪ `minimumAmountOut(slippageTolerance , amountOut?): CurrencyAmount < TOutput >`

Get the minimum amount that must be received from this trade for the given slippage tolerance

Parameters

| Name | Type | Description |
|-------------------|-------------------------|--|
| slippageTolerance | Percent | The tolerance of unfavorable slippage from the execution price of this trade |
| amountOut | CurrencyAmount<TOutput> | - |

Returns

`CurrencyAmount < TOutput >`

The amount out

Defined in

[entities/trade.ts:438](#)

worstExecutionPrice

▪ `worstExecutionPrice(slippageTolerance): Price < TInput , TOutput >`

Return the execution price after accounting for slippage tolerance

Parameters

| Name | Type | Description |
|-------------------|---------|--------------------------------|
| slippageTolerance | Percent | the allowed tolerated slippage |

Returns

`Price < TInput , TOutput >`

The execution price

Defined in

[entities/trade.ts:471](#)

bestTradeExactIn

▪ Static `bestTradeExactIn< TInput , TOutput >(pools , currencyAmountIn , currencyOut , __namedParameters? , currentPools? , nextAmountIn? , bestTrades?): Promise < Trade < TInput , TOutput , EXACT_INPUT >[] >`

Given a list of pools, and a fixed amount in, returns the top `maxNumResults` trades that go from an input token amount to an output token, making at most `maxHops` hops. Note this does not consider aggregation, as routes are linear. It's possible a better route exists by splitting the amount in among multiple routes.

Type parameters

| Name | Type |
|--------|------------------|
| TInput | extends Currency |

| | |
|---------|-------------------------|
| TOutput | extends Currency |
|---------|-------------------------|

Parameters

| Name | Type | Default value | Description |
|-------------------|--|------------------|---|
| pools | Pool[] | undefined | the pools to consider in finding the best trade |
| currencyAmountIn | CurrencyAmount<TInput> | undefined | used in recursion; the original value of the currencyAmountIn parameter |
| currencyOut | TOutput | undefined | the desired currency out |
| __namedParameters | BestTradeOptions | {} | - |
| currentPools | Pool[] | [] | used in recursion; the current list of pools |
| nextAmountIn | CurrencyAmount<Currency> | currencyAmountIn | exact amount of input currency to spend |
| bestTrades | Trade <TInput, TOutput, EXACT_INPUT>[] | [] | used in recursion; the current list of best trades |

Returns

```
Promise < Trade < TInput , TOutput , EXACT_INPUT >[]>
```

The exact in trade

Defined in

[entities/trade.ts:495](#)

bestTradeExactOut

```
▪ Static bestTradeExactOut< TInput , TOutput >( pools , currencyIn , currencyAmountOut , __namedParameters? , currentPools? , nextAmountOut? , bestTrades? ): Promise < Trade < TInput , TOutput , EXACT_OUTPUT >[]>
```

similar to the above method but instead targets a fixed output amount given a list of pools, and a fixed amount out, returns the top `maxNumResults` trades that go from an input token to an output token amount, making at most `maxHops` hops note this does not consider aggregation, as routes are linear. it's possible a better route exists by splitting the amount in among multiple routes.

Type parameters

| Name | Type |
|---------|-------------------------|
| TInput | extends Currency |
| TOutput | extends Currency |

Parameters

| Name | Type | Default value | Description |
|-------------------|---|-------------------|--|
| pools | Pool[] | undefined | the pools to consider in finding the best trade |
| currencyIn | TInput | undefined | the currency to spend |
| currencyAmountOut | CurrencyAmount<TOutput> | undefined | the desired currency amount out |
| __namedParameters | BestTradeOptions | {} | - |
| currentPools | Pool[] | [] | used in recursion; the current list of pools |
| nextAmountOut | CurrencyAmount<Currency> | currencyAmountOut | the exact amount of currency out |
| bestTrades | Trade <TInput, TOutput, EXACT_OUTPUT>[] | [] | used in recursion; the current list of best trades |

Returns

```
Promise < Trade < TInput , TOutput , EXACT_OUTPUT >[]>
```

The exact out trade

Defined in

[entities/trade.ts:576](#)

createUncheckedTrade

- Static `createUncheckedTrade< TInput , TOutput , TTradeType >(constructorArguments): Trade < TInput , TOutput , TTradeType >`

Creates a trade without computing the result of swapping through the route. Useful when you have simulated the trade elsewhere and do not have any tick data

Type parameters

| Name | Type | Description |
|------------|-------------------|---|
| TInput | extends Currency | The input token, either Ether or an ERC-20 |
| TOutput | extends Currency | The output token, either Ether or an ERC-20 |
| TTradeType | extends TradeType | The type of the trade, either exact in or exact out |

Parameters

| Name | Type | Description |
|-----------------------------------|---|---|
| constructorArguments | Object | The arguments passed to the trade constructor |
| constructorArguments.inputAmount | CurrencyAmount<TInput> | - |
| constructorArguments.outputAmount | CurrencyAmount<TOutput> | - |
| constructorArguments.route | Route <TInput, TOutput> | - |
| constructorArguments.tradeType | TTradeType | - |

Returns

`Trade < TInput , TOutput , TTradeType >`

The unchecked trade

Defined in

[entities/trade.ts:346](#)

createUncheckedTradeWithMultipleRoutes

- Static `createUncheckedTradeWithMultipleRoutes< TInput , TOutput , TTradeType >(constructorArguments): Trade < TInput , TOutput , TTradeType >`

Creates a trade without computing the result of swapping through the routes. Useful when you have simulated the trade elsewhere and do not have any tick data

Type parameters

| Name | Type | Description |
|------------|-------------------|---|
| TInput | extends Currency | The input token, either Ether or an ERC-20 |
| TOutput | extends Currency | The output token, either Ether or an ERC-20 |
| TTradeType | extends TradeType | The type of the trade, either exact in or exact out |

Parameters

| Name | Type | Description |
|--------------------------------|--|---|
| constructorArguments | Object | The arguments passed to the trade constructor |
| constructorArguments.routes | { inputAmount: CurrencyAmount<TInput> ; outputAmount: CurrencyAmount<TOutput> ; route: Route <TInput, TOutput> }[] | - |
| constructorArguments.tradeType | TTradeType | - |

Returns

`Trade < TInput , TOutput , TTradeType >`

The unchecked trade

Defined in

[entities/trade.ts:377](#)

exactIn

► Static **exactIn**< TInput , TOutput >(route , amountIn): Promise < [Trade](#) < TInput , TOutput , EXACT_INPUT >>

Constructs an exact in trade with the given amount in and route

Type parameters

| Name | Type | Description |
|---------|------------------|---|
| TInput | extends Currency | The input token, either Ether or an ERC-20 |
| TOutput | extends Currency | The output token, either Ether or an ERC-20 |

Parameters

| Name | Type | Description |
|----------|---|---------------------------------|
| route | Route <TInput, TOutput> | The route of the exact in trade |
| amountIn | CurrencyAmount<TInput> | The amount being passed in |

Returns

Promise < [Trade](#) < TInput , TOutput , EXACT_INPUT >>

The exact in trade

Defined in

[entities/trade.ts:194](#)

exactOut

► Static **exactOut**< TInput , TOutput >(route , amountOut): Promise < [Trade](#) < TInput , TOutput , EXACT_OUTPUT >>

Constructs an exact out trade with the given amount out and route

Type parameters

| Name | Type | Description |
|---------|------------------|---|
| TInput | extends Currency | The input token, either Ether or an ERC-20 |
| TOutput | extends Currency | The output token, either Ether or an ERC-20 |

Parameters

| Name | Type | Description |
|-----------|---|----------------------------------|
| route | Route <TInput, TOutput> | The route of the exact out trade |
| amountOut | CurrencyAmount<TOutput> | The amount returned by the trade |

Returns

Promise < [Trade](#) < TInput , TOutput , EXACT_OUTPUT >>

The exact out trade

Defined in

[entities/trade.ts:209](#)

fromRoute

► Static **fromRoute**< TInput , TOutput , TTTradeType >(route , amount , tradeType): Promise < [Trade](#) < TInput , TOutput , TTTradeType >>

Constructs a trade by simulating swaps through the given route

Type parameters

| Name | Type | Description |
|------------|-------------------|--|
| TInput | extends Currency | The input token, either Ether or an ERC-20. |
| TOutput | extends Currency | The output token, either Ether or an ERC-20. |
| TTradeType | extends TradeType | The type of the trade, either exact in or exact out. |

Parameters

| Name | Type | Description |
|-----------|---|--|
| route | Route <TInput, TOutput> | route to swap through |
| amount | TTradeType extends EXACT_INPUT ? CurrencyAmount<TInput> : CurrencyAmount<TOutput> | the amount specified, either input or output, depending on tradeType |
| tradeType | TTradeType | whether the trade is an exact input or exact output swap |

Returns

```
Promise < Trade < TInput , TOutput , TTradeType >>
```

The route

Defined in

[entities/trade.ts:226](#)

fromRoutes

- Static **fromRoutes**< TInput , TOutput , TTradeType >(routes , tradeType): Promise < [Trade](#) < TInput , TOutput , TTradeType >>

Constructs a trade from routes by simulating swaps

Type parameters

| Name | Type | Description |
|------------|-------------------|--|
| TInput | extends Currency | The input token, either Ether or an ERC-20. |
| TOutput | extends Currency | The output token, either Ether or an ERC-20. |
| TTradeType | extends TradeType | The type of the trade, either exact in or exact out. |

Parameters

| Name | Type | Description |
|-----------|--|---|
| routes | { amount: TTradeType extends EXACT_INPUT ? CurrencyAmount<TInput> : CurrencyAmount<TOutput> ; route: Route <TInput, TOutput> }[] | the routes to swap through and how much of the amount should be routed through each |
| tradeType | TTradeType | whether the trade is an exact input or exact output swap |

Returns

```
Promise < Trade < TInput , TOutput , TTradeType >>
```

The trade

Defined in

[entities/trade.ts:276](#) @uniswap/v3-sdk / [Exports](#) / FeeAmount

Enumeration: FeeAmount

The default factory enabled fee amounts, denominated in hundredths of bips.

Table of contents

Enumeration Members

- [HIGH](#)
- [LOW](#)
- [LOWEST](#)
- [MEDIUM](#)

Enumeration Members

HIGH

- HIGH = 10000

Defined in

[constants.ts:14](#)

LOW

- LOW = 500

Defined in

[constants.ts:12](#)

LOWEST

- LOWEST = 100

Defined in

[constants.ts:11](#)

MEDIUM

- MEDIUM = 3000

Defined in

[constants.ts:13](#) @uniswap/v3-sdk / [Exports](#) / AllowedPermitArguments

Interface: AllowedPermitArguments

Table of contents

Properties

- [expiry](#)
- [nonce](#)
- [r](#)
- [s](#)
- [v](#)

Properties

expiry

- expiry: BigintIsh

Defined in

[selfPermit.ts:19](#)

nonce

- nonce: BigintIsh

Defined in

[selfPermit.ts:18](#)

r

- r: string

Defined in

[selfPermit.ts:16](#)

S

- `s: string`

Defined in

[selfPermit.ts:17](#)

V

- `v: 0 | 1 | 27 | 28`

Defined in

[selfPermit.ts:15](#) @uniswap/v3-sdk / [Exports](#) / BestTradeOptions

Interface: BestTradeOptions

Table of contents

Properties

- [maxHops](#)
- [maxNumResults](#)

Properties

maxHops

- Optional `maxHops: number`

Defined in

[entities/trade.ts:50](#)

maxNumResults

- Optional `maxNumResults: number`

Defined in

[entities/trade.ts:48](#) @uniswap/v3-sdk / [Exports](#) / ClaimOptions

Interface: ClaimOptions

Options to specify when claiming rewards.

Table of contents

Properties

- [amount](#)
- [recipient](#)
- [tokenId](#)

Properties

amount

- Optional `amount: BigintIsh`

The amount of `rewardToken` to claim. 0 claims all.

Defined in

[staker.ts:52](#)

recipient

- **recipient**: string

Address to send rewards to.

Defined in

[staker.ts:47](#)

tokenId

- **tokenId**: BigintIsh

The id of the NFT

Defined in

[staker.ts:42](#) [@uniswap/v3-sdk](#) / [Exports](#) / CollectOptions

Interface: CollectOptions

Table of contents

Properties

- [expectedCurrencyOwed0](#)
- [expectedCurrencyOwed1](#)
- [recipient](#)
- [tokenId](#)

Properties

expectedCurrencyOwed0

- **expectedCurrencyOwed0**: CurrencyAmount < Currency >

Expected value of tokensOwed0, including as-of-yet-unaccounted-for fees/liquidity value to be burned

Defined in

[nonfungiblePositionManager.ts:114](#)

expectedCurrencyOwed1

- **expectedCurrencyOwed1**: CurrencyAmount < Currency >

Expected value of tokensOwed1, including as-of-yet-unaccounted-for fees/liquidity value to be burned

Defined in

[nonfungiblePositionManager.ts:119](#)

recipient

- **recipient**: string

The account that should receive the tokens.

Defined in

[nonfungiblePositionManager.ts:124](#)

tokenId

- **tokenId**: BigintIsh

Indicates the ID of the position to collect for.

Defined in

[nonfungiblePositionManager.ts:109](#) [@uniswap/v3-sdk](#) / [Exports](#) / CommonAddLiquidityOptions

Interface: CommonAddLiquidityOptions

Options for producing the calldata to add liquidity.

Table of contents

Properties

- [deadline](#)
- [slippageTolerance](#)
- [token0Permit](#)
- [token1Permit](#)
- [useNative](#)

Properties

deadline

- **deadline:** `BigintIsh`

When the transaction expires, in epoch seconds.

Defined in

[nonfungiblePositionManager.ts:56](#)

slippageTolerance

- **slippageTolerance:** `Percent`

How much the pool price is allowed to move.

Defined in

[nonfungiblePositionManager.ts:51](#)

token0Permit

- Optional **token0Permit:** [PermitOptions](#)

The optional permit parameters for spending token0

Defined in

[nonfungiblePositionManager.ts:66](#)

token1Permit

- Optional **token1Permit:** [PermitOptions](#)

The optional permit parameters for spending token1

Defined in

[nonfungiblePositionManager.ts:71](#)

useNative

- Optional **useNative:** `NativeCurrency`

Whether to spend ether. If true, one of the pool tokens must be WETH, by default false

Defined in

[nonfungiblePositionManager.ts:61](#) @uniswap/v3-sdk / [Exports](#) / FeeOptions

Interface: FeeOptions

Table of contents

Properties

- [fee](#)
- [recipient](#)

Properties

fee

- **fee**: `Percent`

The percent of the output that will be taken as a fee.

Defined in

[payments.ts:11](#)

recipient

- **recipient**: `string`

The recipient of the fee.

Defined in

[payments.ts:16](#) @uniswap/v3-sdk / Exports / IncentiveKey

Interface: IncentiveKey

Represents a unique staking program.

Table of contents

Properties

- [endTime](#)
- [pool](#)
- [refundee](#)
- [rewardToken](#)
- [startTime](#)

Properties

endTime

- **endTime**: `BigintIsh`

The time that the incentive program ends.

Defined in

[staker.ts:28](#)

pool

- **pool**: [pool](#)

The pool that the staked positions must provide in.

Defined in

[staker.ts:20](#)

refundee

- **refundee**: `string`

The address which receives any remaining reward tokens at `endTime`.

Defined in

[staker.ts:32](#)

rewardToken

- **rewardToken**: [Token](#)

The token rewarded for participating in the staking program.

Defined in

[staker.ts:16](#)

startTime

- `startTime`: `BigintIsh`

The time when the incentive program begins.

Defined in

[staker.ts:24](#) @uniswap/v3-sdk / Exports / IncreaseSpecificOptions

Interface: IncreaseSpecificOptions

Table of contents

Properties

- [tokenId](#)

Properties

tokenId

- `tokenId`: `BigintIsh`

Indicates the ID of the position to increase liquidity for.

Defined in

[nonfungiblePositionManager.ts:41](#) @uniswap/v3-sdk / Exports / MethodParameters

Interface: MethodParameters

Generated method parameters for executing a call.

Table of contents

Properties

- [calldata](#)
- [value](#)

Properties

calldata

- `calldata`: `string`

The hex encoded calldata to perform the given operation

Defined in

[utils/calldata.ts:11](#)

value

- `value`: `string`

The amount of ether (wei) to send in hex.

Defined in

[utils/calldata.ts:15](#) @uniswap/v3-sdk / Exports / MintSpecificOptions

Interface: MintSpecificOptions

Table of contents

Properties

- [createPool](#)
- [recipient](#)

Properties

createPool

- Optional `createPool: boolean`

Creates pool if not initialized before mint.

Defined in

[nonfungiblePositionManager.ts:34](#)

recipient

- `recipient: string`

The account that should receive the minted NFT.

Defined in

[nonfungiblePositionManager.ts:29](#) @uniswap/v3-sdk / [Exports](#) / NFTPermitOptions

Interface: NFTPermitOptions

Table of contents

Properties

- [deadline](#)
- [r](#)
- [s](#)
- [spender](#)
- [v](#)

Properties

deadline

- `deadline: BigintIsh`

Defined in

[nonfungiblePositionManager.ts:131](#)

r

- `r: string`

Defined in

[nonfungiblePositionManager.ts:129](#)

s

- `s: string`

Defined in

[nonfungiblePositionManager.ts:130](#)

spender

- `spender: string`

Defined in

[nonfungiblePositionManager.ts:132](#)

v

- `v: 0 | 1 | 27 | 28`

Defined in

[nonfungiblePositionManager.ts:128](#) @uniswap/v3-sdk / [Exports](#) / QuoteOptions

Interface: QuoteOptions

Optional arguments to send to the quoter.

Table of contents

Properties

- [sqrtPriceLimitX96](#)
- [useQuoterV2](#)

Properties

sqrtPriceLimitX96

- Optional `sqrtPriceLimitX96: Bigintish`

The optional price limit for the trade.

Defined in

[quoter.ts:17](#)

useQuoterV2

- Optional `useQuoterV2: boolean`

The optional quoter interface to use

Defined in

[quoter.ts:22](#) @uniswap/v3-sdk / [Exports](#) / RemoveLiquidityOptions

Interface: RemoveLiquidityOptions

Options for producing the calldata to exit a position.

Table of contents

Properties

- [burnToken](#)
- [collectOptions](#)
- [deadline](#)
- [liquidityPercentage](#)
- [permit](#)
- [slippageTolerance](#)
- [tokenId](#)

Properties

burnToken

- Optional `burnToken: boolean`

Whether the NFT should be burned if the entire position is being exited, by default false.

Defined in

[nonfungiblePositionManager.ts:162](#)

collectOptions

- `collectOptions: Omit < CollectOptions , "tokenId" >`

Parameters to be passed on to collect

Defined in

[nonfungiblePositionManager.ts:172](#)

deadline

- **deadline:** `BigintIsh`

When the transaction expires, in epoch seconds.

Defined in

[nonfungiblePositionManager.ts:157](#)

liquidityPercentage

- **liquidityPercentage:** `Percent`

The percentage of position liquidity to exit.

Defined in

[nonfungiblePositionManager.ts:147](#)

permit

- Optional **permit:** [NFTPermitOptions](#)

The optional permit of the token ID being exited, in case the exit transaction is being sent by an account that does not own the NFT

Defined in

[nonfungiblePositionManager.ts:167](#)

slippageTolerance

- **slippageTolerance:** `Percent`

How much the pool price is allowed to move.

Defined in

[nonfungiblePositionManager.ts:152](#)

tokenId

- **tokenId:** `BigintIsh`

The ID of the token to exit

Defined in

[nonfungiblePositionManager.ts:142](#) [@uniswap/v3-sdk](#) / [Exports](#) / SafeTransferOptions

Interface: SafeTransferOptions

Table of contents

Properties

- [data](#)
- [recipient](#)
- [sender](#)
- [tokenId](#)

Properties

data

- Optional **data:** `string`

The optional parameter that passes data to the `onERC721Received` call for the staker

Defined in

[nonfungiblePositionManager.ts:97](#)

recipient

- **recipient**: string

The account that should receive the NFT.

Defined in

[nonfungiblePositionManager.ts:88](#)

sender

- **sender**: string

The account sending the NFT.

Defined in

[nonfungiblePositionManager.ts:83](#)

tokenId

- **tokenId**: BigintIsh

The id of the token being sent.

Defined in

[nonfungiblePositionManager.ts:93](#) @uniswap/v3-sdk / [Exports](#) / StandardPermitArguments

Interface: StandardPermitArguments

Table of contents

Properties

- [amount](#)
- [deadline](#)
- [r](#)
- [s](#)
- [v](#)

Properties

amount

- **amount**: BigintIsh

Defined in

[selfPermit.ts:10](#)

deadline

- **deadline**: BigintIsh

Defined in

[selfPermit.ts:11](#)

r

- **r**: string

Defined in

[selfPermit.ts:8](#)

s

- **s**: string

Defined in

[selfPermit.ts:9](#)

v
• v: 0 | 1 | 27 | 28

Defined in

[selfPermit.ts:7](#) @uniswap/v3-sdk / [Exports](#) / SwapOptions

Interface: SwapOptions

Options for producing the arguments to send calls to the router.

Table of contents

Properties

- [deadline](#)
- [fee](#)
- [inputTokenPermit](#)
- [recipient](#)
- [slippageTolerance](#)
- [sqrtPriceLimitX96](#)

Properties

deadline

- **deadline**: `BigintIsh`

When the transaction expires, in epoch seconds.

Defined in

[swapRouter.ts:30](#)

fee

- Optional **fee**: [FeeOptions](#)

Optional information for taking a fee on output.

Defined in

[swapRouter.ts:45](#)

inputTokenPermit

- Optional **inputTokenPermit**: [PermitOptions](#)

The optional permit parameters for spending the input.

Defined in

[swapRouter.ts:35](#)

recipient

- **recipient**: `string`

The account that should receive the output.

Defined in

[swapRouter.ts:25](#)

slippageTolerance

- **slippageTolerance**: [Percent](#)

How much the execution price is allowed to move unfavorably from the trade execution price.

Defined in

[swapRouter.ts:20](#)

sqrtPriceLimitX96

- `Optional sqrtPriceLimitX96: BigintIsh`

The optional price limit for the trade.

Defined in

[swapRouter.ts:40](#) `@uniswap/v3-sdk / Exports` / TickConstructorArgs

Interface: TickConstructorArgs

Table of contents

Properties

- [index](#)
- [liquidityGross](#)
- [liquidityNet](#)

Properties

index

- `index: number`

Defined in

[entities/tick.ts:7](#)

liquidityGross

- `liquidityGross: BigintIsh`

Defined in

[entities/tick.ts:8](#)

liquidityNet

- `liquidityNet: BigintIsh`

Defined in

[entities/tick.ts:9](#) `@uniswap/v3-sdk / Exports` / TickDataProvider

Interface: TickDataProvider

Provides information about ticks

Implemented by

- [NoTickDataProvider](#)
- [TickListDataProvider](#)

Table of contents

Methods

- [getTick](#)
- [nextInitializedTickWithinOneWord](#)

Methods

getTick

- `getTick(tick): Promise <{ liquidityNet : BigintIsh }>`

Return information corresponding to a specific tick

Parameters

| Name | Type | Description |
|------|------|-------------|
| | | |

| | | |
|------|--------|------------------|
| tick | number | the tick to load |
|------|--------|------------------|

Returns

`Promise <{ liquidityNet : Bigintish }>`

Defined in

[entities/tickDataProvider.ts:11](#)

nextInitializedTickWithinOneWord

▪ `nextInitializedTickWithinOneWord(tick , lte , tickSpacing): Promise <[number , boolean]>`

Return the next tick that is initialized within a single word

Parameters

| Name | Type | Description |
|-------------|---------|--|
| tick | number | The current tick |
| lte | boolean | Whether the next tick should be lte the current tick |
| tickSpacing | number | The tick spacing of the pool |

Returns

`Promise <[number , boolean]>`

Defined in

[entities/tickDataProvider.ts:19](#) [@uniswap/v3-sdk / Exports](#) / WithdrawOptions

Interface: WithdrawOptions

Options to specify when withdrawing a position.

Table of contents

Properties

- [data](#)
- [owner](#)

Properties

data

- Optional `data: string`

Set when withdrawing. `data` is passed to `safeTransferFrom` when transferring the position from contract back to owner.

Defined in

[staker.ts:66](#)

owner

- `owner: string`

Set when withdrawing. The position will be sent to `owner` on withdraw.

Defined in

[staker.ts:61](#)

id: overview sidebar_position: 1 title: Overview

Table of contents

Enumerations

- [FeeAmount](#)

Classes

- [FullMath](#)
- [LiquidityMath](#)
- [Multicall](#)
- [NoTickDataProvider](#)
- [NonfungiblePositionManager](#)
- [Payments](#)
- [Pool](#)
- [Position](#)
- [PositionLibrary](#)
- [Route](#)
- [SelfPermit](#)
- [SqrtPriceMath](#)
- [Staker](#)
- [SwapMath](#)
- [SwapQuoter](#)
- [SwapRouter](#)
- [Tick](#)
- [TickLibrary](#)
- [TickList](#)
- [TickListDataProvider](#)
- [TickMath](#)
- [Trade](#)

Interfaces

- [AllowedPermitArguments](#)
- [BestTradeOptions](#)
- [ClaimOptions](#)
- [CollectOptions](#)
- [CommonAddLiquidityOptions](#)
- [FeeOptions](#)
- [IncentiveKey](#)
- [IncreaseSpecificOptions](#)
- [MethodParameters](#)
- [MintSpecificOptions](#)
- [NFTPermitOptions](#)
- [QuoteOptions](#)
- [RemoveLiquidityOptions](#)
- [SafeTransferOptions](#)
- [StandardPermitArguments](#)
- [SwapOptions](#)
- [TickConstructorArgs](#)
- [TickDataProvider](#)
- [WithdrawOptions](#)

Type Aliases

- [AddLiquidityOptions](#)
- [FullWithdrawOptions](#)
- [IncreaseOptions](#)
- [MintOptions](#)
- [PermitOptions](#)

Variables

- [ADDRESS_ZERO](#)
- [FACTORY_ADDRESS](#)
- [POOL_INIT_CODE_HASH](#)
- [TICK_SPACINGS](#)

Functions

- [computePoolAddress](#)
- [encodeRouteToPath](#)
- [encodeSqrtRatioX96](#)
- [isSorted](#)
- [maxLiquidityForAmounts](#)
- [mostSignificantBit](#)
- [nearestUsableTick](#)
- [priceToClosestTick](#)
- [subln256](#)

- [tickToPrice](#)
- [toHex](#)
- [tradeComparator](#)

Type Aliases

AddLiquidityOptions

T AddLiquidityOptions: [MintOptions](#) | [IncreaseOptions](#)

Defined in

[nonfungiblePositionManager.ts:77](#)

FullWithdrawOptions

T FullWithdrawOptions: [ClaimOptions](#) & [WithdrawOptions](#)

Defined in

[staker.ts:8](#)

IncreaseOptions

T IncreaseOptions: [CommonAddLiquidityOptions](#) & [IncreaseSpecificOptions](#)

Defined in

[nonfungiblePositionManager.ts:75](#)

MintOptions

T MintOptions: [CommonAddLiquidityOptions](#) & [MintSpecificOptions](#)

Defined in

[nonfungiblePositionManager.ts:74](#)

PermitOptions

T PermitOptions: [StandardPermitArguments](#) | [AllowedPermitArguments](#)

Defined in

[selfPermit.ts:22](#)

Variables

ADDRESS_ZERO

- Const ADDRESS_ZERO: "0x000"

Defined in

[constants.ts:3](#)

FACTORY_ADDRESS

- Const FACTORY_ADDRESS: "0x1F98431c8aD98523631AE4a59f267346ea31F984"

Defined in

[constants.ts:1](#)

POOL_INIT_CODE_HASH

- Const POOL_INIT_CODE_HASH: "0xe34f199b19b2b4f47f68442619d555527d244f78a3297ea89325f843f87b8b54"

Defined in

[constants.ts:5](#)

TICK_SPACINGS

- Const TICK_SPACINGS: { [amount in FeeAmount]: number }

The default factory tick spacings by fee amount.

Defined in

[constants.ts:20](#)

Functions

computePoolAddress

▪ **computePoolAddress(** __namedParameters **): string**

Computes a pool address

Parameters

| Name | Type |
|---|---------------------------|
| __namedParameters | Object |
| __namedParameters.factoryAddress | string |
| __namedParameters.fee | FeeAmount |
| __namedParameters.initCodeHashManualOverride? | string |
| __namedParameters.tokenA | Token |
| __namedParameters.tokenB | Token |

Returns

string

The pool address

Defined in

[utils/computePoolAddress.ts:16](#)

encodeRouteToPath

▪ **encodeRouteToPath(** route , exactOutput **): string**

Converts a route to a hex encoded path

Parameters

| Name | Type | Description |
|-------------|--|---|
| route | Route <Currency, Currency> | the v3 path to convert to an encoded path |
| exactOutput | boolean | whether the route should be encoded in reverse, for making exact output swaps |

Returns

string

Defined in

[utils/encodeRouteToPath.ts:11](#)

encodeSqrtRatioX96

▪ **encodeSqrtRatioX96(** amount1 , amount0 **): JSBI**

Returns the sqrt ratio as a Q64.96 corresponding to a given ratio of amount1 and amount0

Parameters

| Name | Type | Description |
|---------|-----------|---|
| amount1 | BigintIsh | The numerator amount i.e., the amount of token1 |
| amount0 | BigintIsh | The denominator amount i.e., the amount of token0 |

Returns

JSBI

The sqrt ratio

Defined in

[utils/encodeSqrtRatioX96.ts:11](#)

isSorted

• **isSorted**< T >(list , comparator): boolean

Determines if a tick list is sorted

Type parameters

| Name |
|------|
| T |

Parameters

| Name | Type | Description |
|------------|------------------------|----------------|
| list | T[] | The tick list |
| comparator | (a: T, b: T) => number | The comparator |

Returns

boolean

true if sorted

Defined in

[utils/isSorted.ts:7](#)

maxLiquidityForAmounts

• **maxLiquidityForAmounts**(sqrtRatioCurrentX96 , sqrtRatioAX96 , sqrtRatioBX96 , amount0 , amount1 , useFullPrecision): JSBI

Computes the maximum amount of liquidity received for a given amount of token0, token1, and the prices at the tick boundaries.

Parameters

| Name | Type | Description |
|---------------------|-----------|---|
| sqrtRatioCurrentX96 | default | the current price |
| sqrtRatioAX96 | default | price at lower boundary |
| sqrtRatioBX96 | default | price at upper boundary |
| amount0 | BigintIsh | token0 amount |
| amount1 | BigintIsh | token1 amount |
| useFullPrecision | boolean | if false, liquidity will be maximized according to what the router can calculate, not what core can theoretically support |

Returns

JSBI

Defined in

[utils/maxLiquidityForAmounts.ts:68](#)

mostSignificantBit

• **mostSignificantBit**(x): number

Parameters

| Name | Type |
|------|------|
| | |

| | |
|---|---------|
| x | default |
|---|---------|

Returns

number

Defined in

[utils/mostSignificantBit.ts:12](#)

nearestUsableTick

- **nearestUsableTick(tick , tickSpacing): number**

Returns the closest tick that is nearest a given tick and usable for the given tick spacing

Parameters

| Name | Type | Description |
|-------------|--------|-------------------------|
| tick | number | the target tick |
| tickSpacing | number | the spacing of the pool |

Returns

number

Defined in

[utils/nearestUsableTick.ts:9](#)

priceToClosestTick

- **priceToClosestTick(price): number**

Returns the first tick for which the given price is greater than or equal to the tick price

Parameters

| Name | Type | Description |
|-------|---------------------|--|
| price | Price<Token, Token> | for which to return the closest tick that represents a price less than or equal to the input price, i.e. the price of the returned tick is less than or equal to the input price |

Returns

number

Defined in

[utils/priceTickConversions.ts:29](#)

subIn256

- **subIn256(x , y): JSBI**

Parameters

| Name | Type |
|------|---------|
| x | default |
| y | default |

Returns

JSBI

Defined in

[utils/tickLibrary.ts:11](#)

tickToPrice

- **tickToPrice(baseToken , quoteToken , tick): Price < Token , Token >**

Returns a price object corresponding to the input tick and the base/quote token Inputs must be tokens because the address order is used to interpret the price represented by the tick

Parameters

| Name | Type | Description |
|------------|--------|--|
| baseToken | Token | the base token of the price |
| quoteToken | Token | the quote token of the price |
| tick | number | the tick for which to return the price |

Returns

`Price < Token , Token >`

Defined in

[utils/priceTickConversions.ts:14](#)

toHex

▸ `toHex(bigintIsh): string`

Converts a big int to a hex string

Parameters

| Name | Type |
|-----------|-----------|
| bigintIsh | BigintIsh |

Returns

`string`

The hex encoded calldata

Defined in

[utils/calldata.ts:23](#)

tradeComparator

▸ `tradeComparator< TInput , TOutput , TTradeType >(a , b): number`

Trades comparator, an extension of the input output comparator that also considers other dimensions of the trade in ranking them

Type parameters

| Name | Type | Description |
|------------|-------------------|--|
| TInput | extends Currency | The input token, either Ether or an ERC-20 |
| TOutput | extends Currency | The output token, either Ether or an ERC-20 |
| TTradeType | extends TradeType | The trade type, either exact input or exact output |

Parameters

| Name | Type | Description |
|------|---|-----------------------------|
| a | Trade <TInput, TOutput, TTradeType> | The first trade to compare |
| b | Trade <TInput, TOutput, TTradeType> | The second trade to compare |

Returns

`number`

A sorted ordering for two neighboring elements in a trade array

Defined in

[entities/trade.ts:16](#)

id: connect-wallet title: Connecting to Wallets

Introduction

This guide will cover how to connect wallets with `web3-react`. It is based on the [web3-react example](#) found in the Uniswap code examples [repository](#). To run this example, check out the example's [README](#) and follow the setup instructions.

In this example we will walk through setting up `web3-react` and connecting the most popular browser-injected connector, [MetaMask](#), using [@web3-react/metamask](#).

The input parameters to this guide are the chains that we want our app to be able to connect to and their RPC URLs.

The guide will **cover**:

1. Creating a `web3-react Web3ReactProvider`
2. Building a `web3-react InjectedConnector`
3. Connecting and disconnecting the application to the connector

At the end of the guide, we should be able to connect and disconnect your dApp to a MetaMask connector.

For this guide, the following `web3-react` packages are used:

- [@web3-react/core](#)
- [@web3-react/types](#)
- [@web3-react/metamask](#)

:::info This guide uses `web3-react` version 8, which is a beta version. :::

These will be automatically installed by following the example's [README](#).

The core code of this guide can be found in [Web3ContextProvider](#) and [InjectedConnector](#).

Creating a `Web3ReactProvider`

To be able to interact with the methods that `web3-react` offers, we first need to setup a `Web3ReactProvider` and wrap our application in it. `web3-react` uses a [React Context](#) to allow us to use the exposed hooks without additional configuration.

To start, we create a React component called `Web3ContextProvider` in order to wrap the logic of configuring the `Web3ReactProvider`. In this component, we first import `Web3ReactProvider` from [@web3-react/core](#).

The component receives just one prop which is the `children` to which it will be providing the `Context`:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/components/Web3ContextProvider.tsx#L24
```

We then implement the component by rendering the imported `Web3ReactProvider` with the `children` within that:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/components/Web3ContextProvider.tsx#L30-L34
```

Note that we map our list of `Connections` to a *tuple* of the `connector` and `hooks` of the connection. The third element of a `Connection` refers to the `type` of `Connection` being established, which we will later use to keep track of the actively connected wallet.

Finally, having created the `Web3ContextProvider` component, we can navigate to our [index file](#) and wrap the whole application with it:

```
https://github.com/Uniswap/examples/blob/7ac3853bc465aecc428a32be584bbeb833b0a63c/web3-react/src/index.tsx#L16-L22
```

Building an `Injected Connector`

The only parameter that we provided to the `Web3ReactProvider` component is a list of prioritized connectors, declared as `PrioritizedConnectors`. The prioritization ordering is with regards to which connector we want to be active when more than one connector is connected to our application. The list is defined inside our `connectors` module:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/connections.ts#L33-L39
```

Each one of those connectors lives within its own file, and they all follow a similar setup pattern.

An example of a connector in the list is the `InjectedConnector`, which supports wallets that inject an *Ethereum Provider* into the browser window. The most popular example of an injected connector is the *MetaMask* browser extension. To set it up, we import `initializeConnector` function from [core](#) and the `MetaMask` type from [metamask](#):

```
https://github.com/Uniswap/examples/blob/856dbb002e7f38120554ef226f4309c96ce6ea79/web3-react/src/libs/injected.ts#L1-L2
```

We then utilize the templated `initializeConnector` function with `MetaMask` as the type argument:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/injected.ts#L7-L9
```

By passing in `MetaMask` as the type argument, we define the function's required input parameters. In this case, the only parameter we need to pass is an instance of `Metamask`, which receives the `actions` and `onError` parameters. The first parameter defines the actions that `web3-react` performs on its local store for the connector (this usually can be passed through without modification), while the second parameter is the callback invoked when an error occurs.

The return type of the function is a tuple of the initialized `Connector` and the `Hooks` that we can use on it. Using this tuple, we create an instance of a [Connection](#) type, by setting the `type` property to `INJECTED`:

```
https://github.com/Uniswap/examples/blob/856dbb002e7f38120554ef226f4309c96ce6ea79/web3-react/src/libs/injected.ts#L16-L20
```

Finally, we return the instance of `Connection` we created, which is added to the list of prioritized connectors.

:::info For help on creating the rest of the supported connectors of this examples, please visit our [connectors](#) page! :::

Connecting and disconnecting the application to the connector

Having built our `InjectedConnector`, we can now use it in the Context that allows our application to use that connector:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/components/Option.tsx#L5-L11
```

The component receives 5 parameters:

- `isEnabled` determines whether the connector is eligible to be activated
- `isConnected` determines whether the connector is currently active
- `connectionType` determines the `ConnectionType`
- `onActivate` is called once the component has established a connection
- `onDeactivate` is called once the component has disconnected

In the case of *MetaMask*, when declaring the `InjectedConnector` we pass the connector-specific arguments:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/components/ConnectionOptions.tsx#L26-L32
```

Then, in the `html` portion of the `Option`, we can figure out whether we want the current `Option`'s action button to be disabled, and whether clicking the button would result in the connector being connected or disconnected:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/components/Option.tsx#L38-L42
```

Finally, we also have enough information to figure out what action to take when the button is clicked. In the case that the click triggers a connection:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/components/Option.tsx#L38-L42
```

To connect our wallet, all we need to do is to call the `tryActivateConnector` function and pass it the `InjectedConnector`. We then call the `onActivate` callback, which makes the `InjectedConnector` the active connector in our application's state.

`tryActivateConnector` takes as its argument the connector that we want to activate, and attempts to call `activate` on it. If this activation succeeds, it returns the new `ConnectionType`:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/connections.ts#L90-L92
```

In the case that the click triggers a disconnection:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/components/Option.tsx#L29-L36
```

To disconnect, all we need to do is to call `tryDeactivateConnector` and pass in it the `InjectedConnector` we created before. We then call the `onDeactivate` callback, which removes the `InjectedConnector` as the currently active connector from our application's state.

`tryDeactivateConnector` takes as its argument the connector that we want to deactivate, and attempts to call `deactivate` on it. If this deactivation succeeds, it resets the connector's state by calling `resetState` and returns `null`:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/connections.ts#L98-L100
```

Next Steps

Now that we have gone through connecting and disconnecting from an `InjectedConnector`, we will learn how to [connect and disconnect](#) from all the different types of connectors that `web3-react` supports.

id: connectors title: Supported Connectors

Introduction

This guide will cover how to connect our dApp to all the different connectors that `web3-react` supports. It is based on the [web3-react example](#), found in the Uniswap code examples [repository](#). To run this example, check out the examples's [README](#) and follow the setup instructions.

In this example we will cover connecting our dApp to the following connectors:

- Coinbase wallet
- WalletConnect wallet
- Network
- Gnosis safe

:::info For help on setting up `web3-react` and interacting with a MetaMask wallet, please visit our [connecting to wallets](#) page! :::

The input parameters to this guide are the chains that we want our dApp to be able to connect to and their RPC URLs.

The guide will **cover**:

1. Building a Coinbase Wallet connector
2. Building a WalletConnect Wallet connector
3. Building a Network connector
4. Building a Gnosis Safe connector

At the end of the guide, we should be able to connect and disconnect the application to the different connectors listed above.

For this guide, the following `web3-react` packages are used:

- [@web3-react/core](#)
- [@web3-react/types](#)
- [@web3-react/coinbase-wallet](#)
- [@web3-react/walletconnect](#)
- [@web3-react/network](#)
- [@web3-react/gnosis-safe](#)

:::info This guide uses `web3-react` version 8, which is a beta version. :::

The core code of this guide can be found in the top level of our [examples repository](#), under each connectors' name. For example, the code for the Coinbase Wallet connector can be found in the the [coinbase file](#).

Building a Coinbase Wallet connector

The second connector in the list of prioritized connectors that [we provided](#) as a parameter to `Web3ReactProvider` is the *Coinbase Wallet* connector:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/connections.ts#L33-L39
```

To connect to a *Coinbase Wallet* connector, we first need to install [@web3-react/coinbase-wallet](#), as well as [@coinbase/wallet-sdk](#). Having installed the packages, we can import the `CoinbaseWallet` class from [@web3-react/coinbase-wallet](#), as well as the `initializeConnector` function from the [@web3-react/core](#) package:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/coinbase.ts#L1-L2
```

We can now build our connector, supplying the required arguments:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/coinbase.ts#L8-L19
```

We pass `CoinbaseWallet` as the type argument to `initializeConnector`'s templated parameter. Similar to the case of the `InjectedConnector`, the `CoinbaseWallet` class is a class that extends the `AbstractConnector` class, which is part of the [@web3-react/core](#) package. The parameter provided to `initializeConnector` is a function that receives an `actions` object, and expects an instance of `CoinbaseWallet` (to match the type argument) to be returned.

We build the new `CoinbaseWallet` instance by passing the `actions` object, an `options` object, and an `onError` callback. `onError` handles errors that occur during interaction with the connector, and `options` is used to configure the connector. In our case, we pass the `url`, `appName` and `reloadOnDisconnect` options: `url` is the *RPC URL* to connect to that was provided as an argument to the example application, `appName` is the name of our application, and `reloadOnDisconnect` is a `boolean` that indicates whether the application should reload when the user disconnects from the wallet.

After building the connector, we use its two return types, the `Connector` and it's respective hooks, and build a `Connection` object by setting the connection's type as the *Coinbase wallet*:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/coinbase.ts#L20-L24
```

Having built the connector, all that remains is to build the user interface and supply it to our [ConnectionOptions](#) component, just as we did with the `InjectedConnector`:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/components/ConnectionOptions.tsx#L39-L46
```

Building a WalletConnect Wallet connector

The third connector in the list of prioritized connectors that we provided to [Web3ReactProvider](#) is the *WalletConnect Wallet connector*.

To connect to a *WalletConnect Wallet* connector, we first need to install [@web3-react/walletconnect](#), as well as [@walletconnect/ethereum-provider](#). Having installed the packages, we can import the `WalletConnect` class from [@web3-react/walletconnect](#), as well as the `initializeConnector` function from [@web3-react/core](#) package:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/wallet-connect.ts#L1-L2
```

We can now build our connector, supplying the required arguments:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/wallet-connect.ts#L8-L17
```

The main difference from the *Coinbase Wallet* connector lies in the arguments that the `WalletConnect` class requires to be instantiated. `web3-react` knows about this difference, as we passed the type argument `WalletConnect` to `initializeConnector`, thus specializing the type of `AbstractConnector`. In this case, the class receives three arguments, including `actions` and `onError`, identical to those supplied in the *Coinbase Wallet* connector case.

The difference lies in the second argument, which is an `options` object. In this case, we are passing the `rpc` parameter, which is an object that maps the chain ID to the *RPC URL* to connect to. We have already created this `map` in our [constants](#) file using our example's parameters. The other option that we are passing is the `qrcode`, which is a `boolean` that indicates whether the QR code should be displayed in the browser. In our case, we are passing `true` as we want to show the QR code.

Having built the connector, we just need to build the user interface to enable user interaction with the connector, and supply it to our `ConnectionOptions`:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/components/ConnectionOptions.tsx#L49-L56
```

Building a Network connector

The `Network connector`, alongside the `Gnosis Safe connector`, are two of the connectors that we do not surface through our user interface, but instead we connect to them programmatically. In contrast to the previous `Connectors`, these do not come with any pre-built user interface for the user to interact with. We attempt to connect to them **eagerly** in our `Web3ContextProvider` component through a hook:

```
https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5f34fc9f98f80b0/web3-react/src/libs/components/Web3ContextProvider.tsx#L9-L13
```

The `useEagerlyConnect` hook is called in the `Web3ContextProvider` component and attempts to connect to the Network Connector and the Gnosis Safe Connector. The hook is named **eagerly** as it is called in the component's body as `React effect` when the component is first rendered. In the hook implementation we attempt to call `web3-react`'s `connectEagerly` function if it exists on the connector, otherwise we call `activate` otherwise. The `connectEagerly` function attempts to connect our application to the connector, and **fails silently** if it does not succeed:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/hooks.ts#L15-L19
```

Before eagerly connecting, we first need to initialize the connectors. We start by building the Network connector, and we first need to install `@web3-react/network`, and import the `Network` class from it. Note how this Connect does not require any package besides its `web3-react` package to function. We also need to import the `initializeConnector` function from `@web3-react/core`:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/network.ts#L1-L2
```

We can now build our connector, supplying the required arguments:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/network.ts#L8-L15
```

The main difference from the other connectors lies in the arguments that the `Network` class requires to be instantiated. `web3-react` knows about this difference, as we passed the type argument `Network` to `initializeConnector`, thus specializing the type of `AbstractConnector`. In this case, the class receives `actions`, which is identical to that supplied in the rest of the connectors; `urlMap`, which is an object that maps the chain ID to the RPC URL to connect to, which we have already created in our `constants` file; and `defaultChainId` which is the chain ID to connect to by default.

After building, the connector, we can create a `Connection` instance by supplying it the return value of the `initializeConnector` function, and the `Network` class:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/network.ts#L16-L20
```

All that remains is to return the constructed `Connection` instance.

Building a Gnosis Safe connector

Similar to the Network connector, we build the Gnosis Safe connector. We start by first installing `@web3-react/gnosis-safe`, and import the `GnosisSafe` class from it. We also need to import the `initializeConnector` function from `@web3-react/core`:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/gnosis.tsx#L1-L2
```

The Gnosis Safe connector is the simplest of them all, as it does not require any additional parameterization other than `actions`:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/gnosis.tsx#L6-L9
```

Having initialized the connector, we can now build the `Connection` instance and return it:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/gnosis.tsx#L10-L14
```

Next steps

Now that we have gone through building all of the different types of supported connectors, we will learn how to [switch chains](#).--- id: switch-chains title: Switching Chains

Introduction

This guide will cover how to prompt a wallet that has connected to our dApp to switch chains using `web3-react`. It is based on the [web3-react example](#), found in the Uniswap code examples [repository](#). To run this example, check out the examples's [README](#) and follow the setup instructions.

:::info For help on setting up `web3-react` and interacting with a MetaMask wallet, please visit our [connecting to wallets](#) page! :::

The input parameters to this guide are the chains that we want our dApp to be able to connect to and their RPC URLs.

At the end of the guide, we should be able to switch chains on the connected wallet.

For this guide, the following `web3-react` packages are used:

- [@web3-react/core](#)

:::info This guide uses `web3-react` version 8, which is a beta version. :::

The core code of this guide can be found in [connections](#).

Switching Chains

Having [setup our application](#) to use `web3-react` and having built out the ability to [connect and disconnect wallets](#), we can now move on to switching chains.

Switching chains requires two parameters, the `chainId` we want to switch to, and the current `connectionType`:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/connections.ts#L64
```

Given the `ConnectionType`, we can retrieve the actual connector:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/connections.ts#L69
```

Then, depending on the `ConnectionType`, we determine how to switch chains. For the `Network` or `WalletConnect` cases, we call `web3-react`'s `activate` function with the supplied `chainId`:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/connections.ts#L71-L74
```

The rest of the connectors require us to build an `AddEthereumChainParameter` object and pass it to the `web3-react`'s `activate` function:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/connections.ts#L77-L84
```

The metadata required to build `AddEthereumChainParameter` are defined in our constants file:

```
https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9c5c/web3-react/src/libs/constants.ts#L27-L40
```

Next steps

Know you know how to support `web3-react`'s most common use cases! Stay tuned for follow up guides.

id: overview sidebar_position: 1 title: Overview

web3-react

Welcome to `web3-react` !

`web3-react` provides abstractions to assist you with connecting your dApp to web3 connectors and exposes methods to interact with those connections. It currently supports connecting to the following wallets:

- Network
- Injected wallets (eg MetaMask)
- Gnosis safe
- Coinbase wallet
- WalletConnect wallet

To begin, we recommend looking at our [guides](#) which include [runnable examples](#) and walkthroughs of core usages. These guides will help you better understand how to use `web3-react` and integrate it into your application.

:::info This guide uses `web3-react` version 8, which is a beta version. :::

Installation

`web3-react` consists of many packages, each providing different functionalities. The [core](#) package exposes the methods used to interact with web3 connectors, the [types](#) package declares useful types, while the others are installed to enable interactions with different connectors.

To interact with `web3-react` we recommend installing though npm:

```
npm install --save @web3-react/core
```

or yarn:

```
yarn add @web3-react/core
```

Developer Links

- [web3-react on Github](#)

[npm@latest v8.2.0](#) [minzipped size 24.1 kB](#) [discord](#) [join chat](#)